A distributed data-mining software platform for extreme data across the compute continuum

D2.2 First Release of the EXTRACT Data Infrastructure and Data Mining Framework

Version 1.0

Documentation Information

Contract Number	101093110
Project Website	www.extract-project.eu
Contractual Deadline	M15, 31st March 2024
Dissemination Level	PU
Nature	R
Author	IBM
Contributors	BIN, BSC, MATH, SIX, URV, LRI
Reviewer	URV
Keywords	Data, Mining, Infrastructure

Change Log

Version	Description Change
V0.1	IBM: Initial Draft and structure
V0.2	URV, IBM, SIX , LRI content
V0.3	IBM - model serving demonstrator
V0.9	Review Acceptance from URV
V1.0	Final review IBM. Ready for submission

The EXTRACT Project has received funding from the European Union's Horizon Europe programme under grant agreement number 101093110.

Table of Contents

1. Introduction

This deliverable marks the first release of the Data Infrastructure and Data Mining of EXTRACT, as shown in Figure 1.1 below. Efficient large-scale data storage and processing across the compute continuum is a key property of EXTRACT, conforming to the Extreme Data trait of this type of projects.

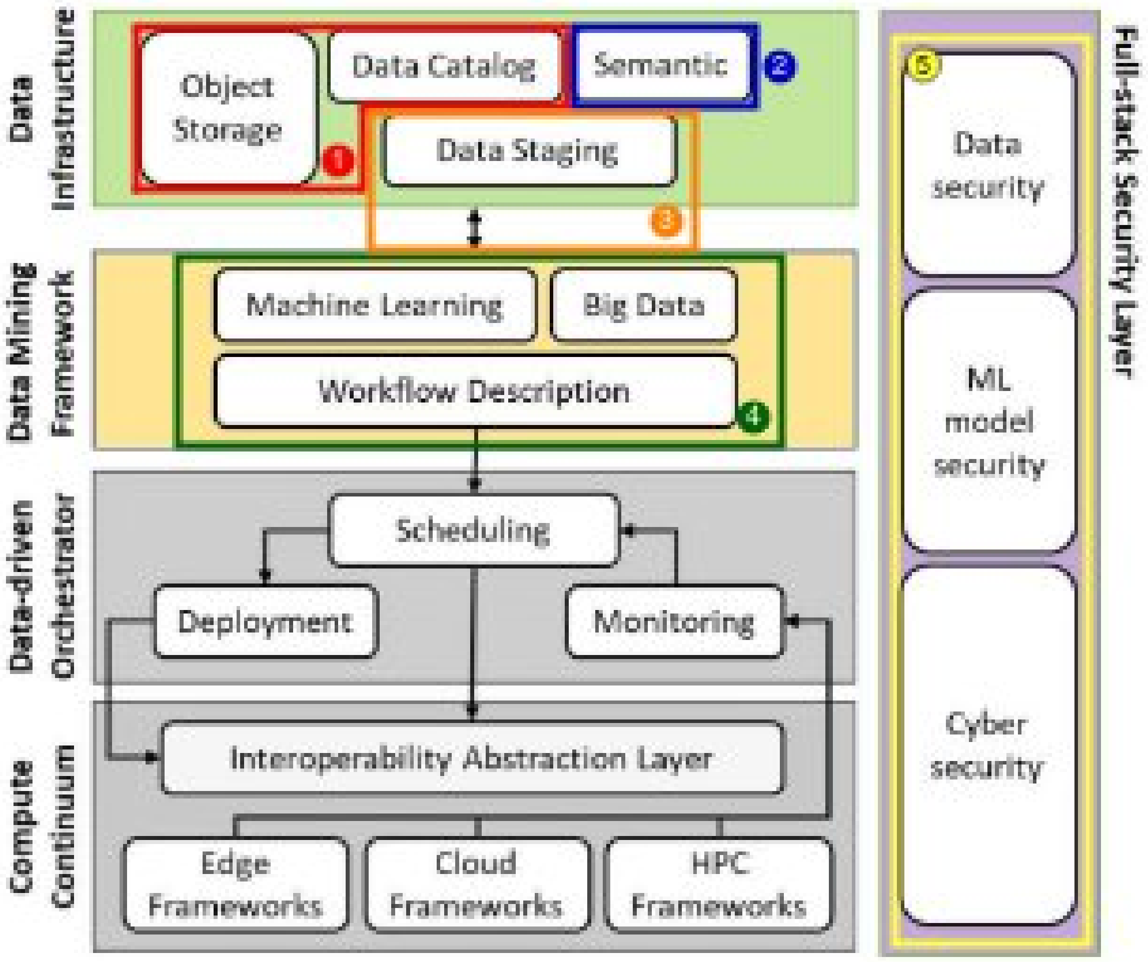

Figure 1.1 Data Infrastructure and Data Mining Layers in EXTRACT Architecture

The EXTRACT data infrastructure platform revolves around distributed data lake and data processing. It consists of several key components, identified already in D2.1 as matching the project's requirements. Hence, this document will unfold starting with an architectural overview of data infrastructure and then diving into each component's function and value.

A similar focus in this document is provided for the Data Mining layer of EXTRACT (T2.3). This layer deals with the higher, semantic level of data processing workflows that EXTRACT applications may require, combining big data elastic ETL, model training/inference and data set generation/ingestion.

Finally, we discuss the subset of functionality that is selected for a first Minimum Viable Product (MVP) of EXTRACT. This section further includes technical descriptions of MVP demonstrators for WP2.

1.1. Purpose and objectives

Key purposes and objectives include:

- **Showcasing Progress**: Highlight the concrete steps taken from the project's previous milestone in M6 to reach the current state of the MVP, detailing the technological, architectural, and operational advancements made.
- **Defining Interim Results**: Clearly distinguish between the ultimate goals of the compute continuum infrastructure and the partial achievements represented by the MVP. This involves outlining the specific functionalities, performance metrics, and capabilities enabled by the MVP, as well as the gaps or areas for further development.
- **Setting the Stage for Future Development**: Use the MVP as a benchmark for future iterations, identifying both the successes and shortcomings of the current approach. This sets clear expectations for the project's next phases, including enhancements, expansion of capabilities, and integration of additional components into the continuum.
- **Aligning with Functional and non-Functional Objectives**: Ensure that the progress and lessons learned from the MVP are aligned with the project's overarching goals. This includes improving efficiency, reducing latency, enhancing data processing capabilities, and fostering a more adaptable and resilient computing infrastructure, while ensuring that we address non-functional requirements such as security matters.

1.2. Relationship with other WPs and Previous Deliverables

Deliverable	Task	Relation
D1.2	T1.2	First release of the EXTRACT use-cases
D2.1	T2.1	First release of the EXTRACT data infrastructure and data mining framework
D3.1	T3.1	First release of the data-driven orchestration and monitoring
D4.1	T4.1	Compute Continuum Specification and First Integration Plan

Table 1. Relationship with other WPs

1.3. Document structure

This document is organized in several sections:

- Section 1 is the introduction part of the document.
- Section 2 reviews the Data Infrastructure
- Section 3 reviews the Data Mining layer
- Section 4 discusses aspects of data security, privacy and integrity
- Section 5 presents the first MVP and a related demonstrator

- Section 6 details early evaluations of components
- Section 7 is a short conclusion

The document concludes by listing the acronyms, abbreviations and bibliography references.

2. Data Infrastructure

Presented below is an overview of the entire data management section in the EXTRACT platform. It includes three main work areas or layers (corresponding to different tasks in WP2, relating to Figure 1.1 above): the data infrastructure (T2.2), the data mining layer (T2.3), and the data security and privacy module (T2.4).

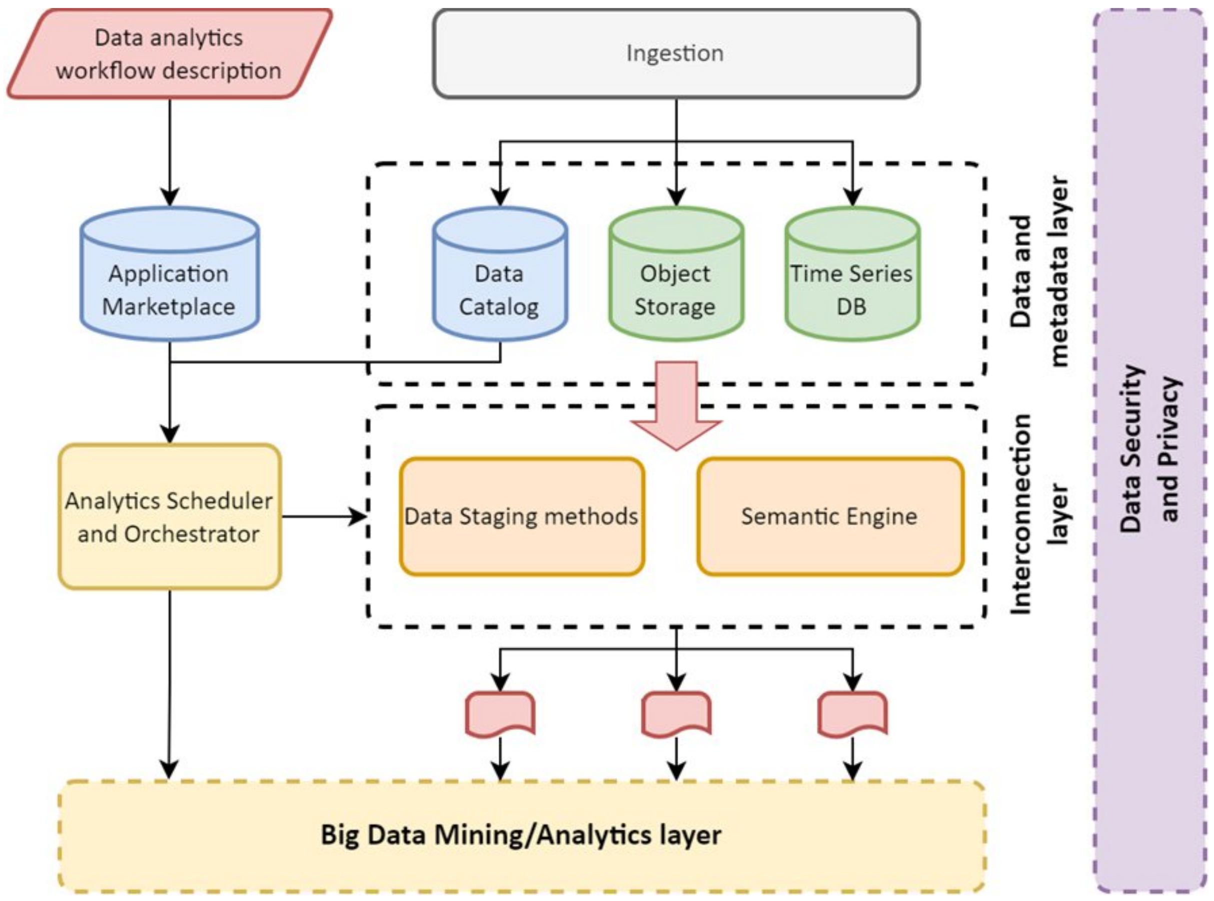

Figure 2.1 EXTRACT Data Management: Data Infrastructure, Data Mining and Data Security & Privacy

Figure 2.1 shows an overall view of the components within these areas and their main interactions as presented in D2.1. In what follows, we will present a description of the different software components that constitute the data handling section of EXTRACT. In general, the components are organised into the following, more specific, layers:

- **Ingestion:** This is a process responsible for inserting the data into the EXTRACT platform. It includes the transfer of data into the data layer and the associated pre-processing tasks necessary, which include metadata generation and extraction.
- **Data and metadata layer:** This layer includes all software solutions that store data in the platform. The data layer is essentially a data lake that includes space

for bulk data in Object Storage and time-based information in a Time Series DB. The Data Catalog stores all the metadata relative to the data in store and provides the necessary means for data discovery and searchability. Metadata includes information of two types: that relevant for data identification (names, origin, history, etc.) and that relevant for the applications that process it (indexes, metrics, methods or procedures, etc.).

- **Semantic layer:** Provides a smart way to integrate and retrieve data from the data layer by creating and maintaining logical relations between different pieces of stored data. It is tightly coupled with the domain of the application using the data.
- **Data staging layer:** This layer manages a scalable solution that provides elastic methods to effectively prepare data stored in the data lake for the data mining frameworks, where it will be mainly processed. This includes tasks such as data partitioning, filtering, or transformation (such as ETLs). The objective of this layer is to provide staging in an elastic, data-driven, and smart fashion, meaning that compute resources destined to staging data dynamically match the requirements of mining applications and their input data volume.
- **Data mining layer - framework integration and workflow definition:** This layer manages the main data processing aspect in the EXTRACT platform. It deals with the overall method for workflow definition and the use and interaction of the different tools and frameworks for data mining.
- **Data security, privacy, and integrity:** security is transversal to the the EXTRACT platform. This layer includes tools and mechanisms to secure data, its privacy, and its integrity across all the different components in the platform.

The data infrastructure components (such as data staging methods and semantic engine) together with the data mining components (analytics and ML frameworks) will be utilised by EXTRACT workflows through the workflow orchestrator described in D3.2. Additionally, the compute substrate where all these components run will be managed seamlessly over the continuum as described in D4.2.

2.1. Data & Metadata Layer

Object Storage

As outlined in D2.1, the adoption of S3-compatible object storage as the central data backbone fulfils the comprehensive data handling requirements in the project. That makes S3-compatible storage the best option for massive data sets. Organising data as objects in flat rather than traditional file hierarchies, ensures a simplistic and scalable access to data in distributed settings. It fully complies with the explicit requirements of the project. Object storage provides data integrity, and allows simple geographical data distribution, meeting key Extract data needs such as high availability, scalability, and security as identified in D2.1.

Time series DB

Time series databases (TSDBs) are specialized database systems optimized for storing and managing time-stamped data. Such data can represent measurements, events, or observations sequentially ordered by time, making TSDBs particularly suited for a wide range of applications, from financial market data analysis to IoT device telemetry and environmental monitoring.Time series databases play a crucial role in handling dynamic, continuously evolving data that is inherently temporal in nature.

Key Features of Time Series Databases:

- **Efficient Storage**: TSDBs are designed to store large volumes of time-stamped data efficiently, using compression algorithms and data structures optimized for time-based querying.High Throughput: They can handle high write and read throughput, accommodating the data velocity typical in real-time monitoring systems and IoT applications.
- **Time-Based Queries**: TSDBs support queries that are time-centric, such as aggregating data over specific time intervals, computing moving averages, or finding time-based patterns.
- **Data Retention Policies**: They often offer automated data retention policies, allowing for the ageing out of old data to manage storage requirements actively.

Data Catalog

A second integral part of the EXTRACT data backbone is the global metadata catalog. This catalog leverages the positive attributes of S3-based services and introduces a comprehensive global management system for metadata. The goal is to enhance the efficiency of search functionalities across different service providers. In terms of implementation, the model consists of three core resources (Figure 2.1.1):

1. data-object: This resource acts as a proxy for data stored in an S3 bucket/object from a specific provider. It manages the lifecycle of S3 objects, simplifying data upload and download processes.
2. data-record: This resource allows users to add additional, user-specified metadata for an object. Enabling the attachment of rich, domain-specific metadata to objects enhances the precision of searching for relevant data.
3. data-set: This resource defines dynamic collections of data-object and/or data-record resources through filters. Administrators, managers, or users can define these collections, providing a flexible and customizable approach to data organization.

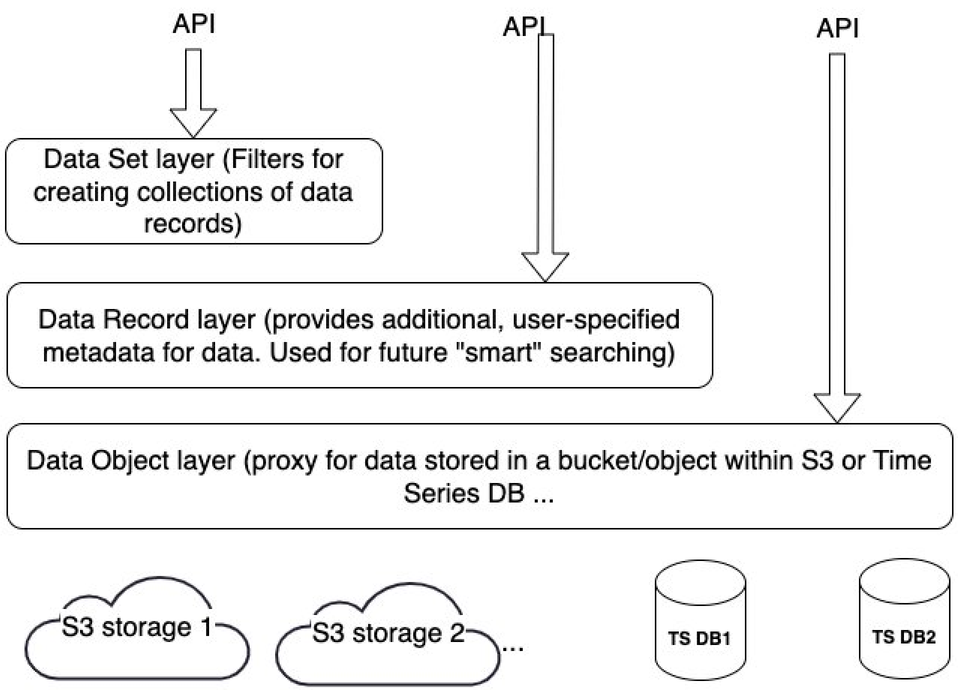
API
API
API
Data Set layer (Filters for creating collections of data records)
Data Record layer (provides additional, user-specified metadata for data. Used for future "smart" searching)
Data Object layer (proxy for data stored in a bucket/object within S3 or Time Series DB ...
S3 storage 1
S3 storage 2
...
TS DB1
TS DB2

Figure 2.1.1: Data Object, Data Record and Data Set layers

Collectively, these resources establish a versatile data management framework applicable to EXTRACT use-cases. The typical workflow involves creating a data-object (implicitly creating the S3 object), optionally adding metadata using a data-record object, and finally, finding and using the relevant data-object resources included in a data set. Nuvla facilitates the "using" element by binding data types to user applications capable of processing the data, offering seamless integration between data management and application utilization.

2.2. Semantic Layer

The Semantic Layer, an integral component of the EXTRACT framework, aims to enrich and contextualize the data harvested from diverse sources, thereby facilitating a more nuanced and comprehensive analysis. By leveraging advanced semantic technologies and ontologies, this layer provides a structured and meaningful representation of data, which is pivotal for achieving nuanced insights and actionable intelligence in real-time scenarios.

2.2.1. Ontology-Driven Data Integration

At the heart of the Semantic Layer is the utilization of ontologies - structured sets of terms and concepts representing the domain knowledge. These ontologies serve as a model backbone for integrating heterogeneous data, ensuring that information from disparate sources is harmoniously blended and semantically enriched. This integration not only encompasses data harmonization but also involves inferring new knowledge by exploiting the relationships and rules defined within the ontologies.

2.2.2. Semantic Annotation and Reasoning

To further augment the data with semantic depth, the layer employs semantic annotation processes. These processes tag data elements with ontology-defined concepts, effectively binding them with domain-specific meanings. Coupled with semantic reasoning, this enriched data undergoes a layer of logical inference, uncovering implicit relationships and deducing new facts that were not directly observable from the raw data. This annotated data is then enriched through logical inference enabled by reasoning engines like Pellet and HermiT, uncovering hidden relationships and deducing new facts. The Virtuoso triplestore acts as a robust repository for this semantically rich data, supporting efficient storage and SPARQL querying capabilities. Standards such as RDF, RDFS, and OWL provide the foundational framework for data modelling and representation, ensuring uniformity and comprehensibility across the semantic web. Additionally, the RDFLib Python library offers versatile tools for transforming and manipulating data into RDF format, enabling seamless integration and application development within this enriched semantic ecosystem. Together, these technologies form a powerful infrastructure for enhancing data with semantic depth, fostering advanced analysis, and facilitating knowledge discovery.

2.2.3. Support for Dynamic and Extreme Data

Given the project's focus on managing extreme data scenarios, such as those presented by personalized evacuation routes in crisis situations, the Semantic Layer's design is inherently dynamic. In particular, the ontology implemented is able to manage dynamic information with their time of arrival (timestamp).

It is also capable of adapting to the rapid influx of high-volume, heterogeneous data, ensuring that the semantic enrichment processes scale efficiently and remain responsive to the evolving data landscape. This is ensured by the ontology design and the adoption of scalable triple stores.

2.2.4. Facilitating Advanced Data Analysis

By providing a semantically enriched and unified view of the data, the Semantic Layer significantly enhances the analytical capabilities of the EXTRACT platform. Analysts and data scientists can leverage this unified view to perform complex queries and analyses that span across multiple data sources, extracting insights that would be challenging to obtain from non-semantic, siloed data. Furthermore, this semantic foundation enables the application of advanced AI and machine learning algorithms, driving innovative solutions tailored to the specific needs of crisis management scenarios.

2.2.5. Infrastructure Optimization and Performance

The Semantic Layer is optimized for high performance and scalability. It employs state-of-the-art semantic indexing and caching techniques to ensure that the semantic operations — from data integration to reasoning — are performed with minimal latency, thus supporting the real-time requirements of the EXTRACT platform. This emphasis on performance optimization ensures that the system remains robust and effective, even in the face of data-intensive challenges presented by extreme data scenarios.

2.3. Data Staging

The data staging layer is the link between the data storage components in EXTRACT and the data mining layer. Its objective is to prepare the data for its easy consumption by the data processing applications by applying simple data management logic and data partitioning. It has two key goals: data preparation and automatic data-driven resource provisioning.

For data preparation, the data staging layer must be able to support different data transformations required by the mining layer. This includes tasks like filtering, small aggregations, or combinations, and typical ETL operations. However, the most important task is data partitioning, which will allow the EXTRACT platform to support big data sets (Extreme data) by splitting the load into multiple, parallel workers.

On the other hand, to apply these data operations effectively on highly variable Extreme data, it is necessary to rely on a computing substrate that is able to quickly match such varying demands. To this end, the EXTRACT data staging layer includes a new tool for the smart data-driven provisioning of resources that is able to scale resources dynamically to match the needs of the different applications based on the demands and data volume of each particular execution.

In this document we report our first exploration and implementation of such a smart provisioning tool, which we evaluated against one of the project's use cases (Section 6.1): the data intensive processing of TASKA use case C.

2.3.1. Lithops data-driven smart provisioning

To find the right number of workers to perform a distributed data processing task is hard. Typically, analytics frameworks split the workload by chunking data in a predefined partition size (a global chunk size) and create as many subtasks as happen to be needed. However, this is usually suboptimal for most applications. Optimal problem partitioning usually depends on the data volume, but also on the characteristics of the process at hand and its particular requirements. Importantly, the compute resources underneath, and hence the available workers that run the tasks, may behave differently for each process they run, meaning that the amount of data they are capable of processing varies from task to task. This decision increases in complexity when dealing with extreme data, not only because of the huge volumes of data to manage, but because of the variability of it, which requires a fast and optimal configuration of resources.

With the aim of simplifying these decisions, we study and prototype a data-driven smart provisioning layer on top of the Lithops tool-kit to enable dynamic and intelligent decision-making in worker provisioning, customised to the demands of extreme-data applications. In the world of data-intensive computing, clearly exemplified within Extract by the TASKA C use-case, the ability to efficiently process vast amounts of data is really important. This efficiency is not only about handling the data, but doing it in a manner that is both time and cost-effective.

Taking a step back, to scale a process in a single machine, you would typically scale the machine vertically. Also known as scaling up, it refers to increasing computing power by using a bigger machine. Since machines have a limit to how big they can be, you may reach the point where you scale horizontally. Also known as scaling out, this refers to increasing computing power by utilising more machines and distributing the workload along them, each processing a slice of the problem. Each strategy offers different advantages and inconveniences, and it is possible to combine them, creating a complex trade-off. In the case of scaling out, partitioning input data has a computational overhead and requires specialised tools to read data in diverse formats, on the other hand, scaling up may not be always available, become difficult to manage, and it is also possible that the task does not completely utilise the resources well or the overhead of managing work within such a big process becomes too big.

The Lithops data-driven smart provisioning layer aims to navigate these trade-offs by leveraging real-time data and application-specific requirements to make informed decisions about the optimal scaling strategy. Whether it is determining the appropriate size of workers or the number of workers needed, this intelligent layer addresses the problem by assessing the current context to optimise resource allocation. This is done to enhance performance and cost-effectiveness of the data processing tasks.

2.3.2. Smart decisions for compute scaling and provisioning

To find out what is the best approach (in terms of number and size of workers), for each data-processing task we must take a decision. Lithops will use different sources of information to keep these decisions informed and effective.

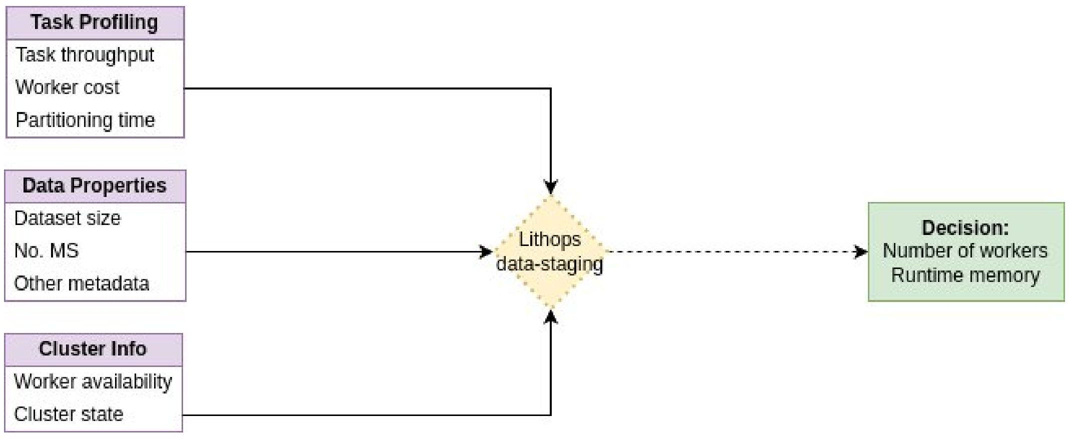

Figure 2.3.1: Lithops smart-provisioning layer decisions.

Figure 2.3.1 provides an overview of the decisions the Lithops smart data-staging layer has to make as well as the inputs considered. The Lithops data-staging layer should make a decision, which is the number of workers as well as the runtime memory to process the data. The objective of the Lithops data-staging layer is to adapt the provisioning of resources to the specific requirements of the computational process.

The goal of the smart data-staging layer is to enable a pipeline or workflow to make better decisions that minimise some objective function based on input variables, much like an optimisation problem. The goal is to minimise the objective function (which is execution time, cost) based on the previously defined inputs.

The smart data-staging component has to make a decision for each process that needs it, and executes before it happens to choose the optimal configuration, this is because pipelines/workflows have intermediate steps whose requirements are not known beforehand, and their data and computational requirements vary from each other.

The inputs to make the decision can be seen in Figure 2.3.1, those are the cluster information, which gives information of the current cluster state and available computing, the data properties, which is the data to be processed by the pipeline/workflow and the task profiling, which is created by previous executions and gives an idea of the scalability of a given process. A more detailed list of those items that Lithops should consider to make decisions are:

- Data volume and format (input dataset)
- Cluster capacity / available worker configurations
- Task worker throughput (MB/s) in a particular configuration (e.g., 1 CPU, 1GB RAM) and other performance metrics (from a history of executions)
- Cost of workers based on their configuration (either monetary or resource occupation)
- Time and cost of partitioning the data format
- Ingestion time for a particular worker configuration
- Deadline / time objectives

The cluster capacity and available worker configurations represent the current state of the cluster, this will aid the decision making process and limit its search space. Task worker throughput is a type of dynamic input which depends on previous executions, it changes and converges to a value with more executions. Time and cost of partitioning is the time it takes to partition a given dataset in different chunks, ingestion time is the time it takes to process a given chunk/dataset for a particular step and worker configuration and finally, deadline/time objectives is the deadline for a given step to process the data, all these parameters affect the decision-making of the smart data-staging layer.

Some of these inputs are static, meaning that they are not refined and subject to change, and some are dynamic, which can be updated across multiple executions. An example of this is the task profile, where the throughput converges to a value the more data from different executions we have.

Finally, the decision-making process evolves with each pipeline execution, it relies on the continuous feedback loop to use the outcomes of past executions to inform future executions and make them cost and execution time effective.

2.3.3. Smart provisioning tool API

The Lithops data-driven smart provisioning layer aims to create a set of functionalities that enable dynamic decision-making for optimal runtime memory and chunk size based on the specific requirements of each task. The goal is to make this as transparent as possible for the user, and thus its utilisation will be integrated into the Lithops existing API. In particular, smart provisioning will be applied to calls to the Lithops "map" primitive, taking the function (task) and input data to make its decisions on the most efficient worker configuration.

Internal to Lithops, the smart provisioning API is designed to integrate into the Lithops framework as a plug-in, offering a mechanism to analyse, predict and apply the most suitable configuration for distributed tasks. This integration is important for enhancing the efficiency and scalability of data-extreme applications, especially those that are subject to the complexities of processing extreme data volumes under varying computational demand.

Lithops will only need to feed the data and task properties and the cluster information to the smart provisioning tool before each execution. By understanding the specific demands of each task, the API can accurately forecast the resources required to execute the task efficiently.

The provisioning tool leverages a database of historical performance metrics and current system state information, including available cluster capacities and worker configurations. This data is critical for the decision-making process, allowing it to consider past performance trends and current resource availability in its calculations. By correlating this information with the task's requirements, the tool can identify the most cost-effective configuration.

Once the optimal configuration is determined, the tool returns to Lithops the optimal plan of data partitioning or distribution so that Lithops can provide and run the correct worker configuration. This method makes sure that each job is provisioned with the ideal worker configuration and parallelism. Since it is automatically applied to Lithops executions, it significantly reduces the operational cost of managing the scale and configuration (right-sizing) manually for each application.

2.3.4. Next steps

At this point, we have achieved a new tool for smart resource provisioning for data staging tasks in data processing workflows. However, its applicability and ease of use is still limited and requires further work to achieve the objectives of EXTRACT. For instance, the decision tool requires a manual task profiling run beforehand that feeds the automatic scaling decisions. Also, the tool only applies a general optimization of cost and time that tries to minimise both variables, but it does not take into account other possible restrictions such as execution time objectives that may be crucial for time-constrained applications. In the remainder of the project, we will keep working on these lines to explore solutions to these problems by evolving Lithops and the data-driven smart provisioning tool.

In summary, we plan to enhance Lithops with the integration of advanced resource usage monitoring and automatic task profiling capabilities, as well as introducing the ability to specify an execution deadline for task execution, which affects the scaling decision made by the smart data-staging layer. The objective is to further automate the optimization of resource allocation. This way, we aim to eliminate the need for manually trying different configurations, and systematically explore configurations for a better, and less involved, exploration that yields better execution efficiency.

In more detail, we will first work towards adding resource usage and monitoring to capture detailed metrics on CPU, memory, storage and network utilisation for each task executed with the Lithops framework. This will work in combination with available system monitoring resources and will provide a standard way to characterise tasks to then create profiles that the smart provisioning tool can leverage.

Next, we plan to build automatic profiling into Lithops to simplify the task of recollecting this information automatically. Further, this automatic process will keep updating task profiles as tasks are run to further improve their characterisation and, therefore, the accuracy of the smart decisions.

By enabling users to specify task execution deadlines, we can further influence the decision tool for a more precise provisioning. A time objective means that some of the available configurations should be discarded early if they cannot meet it. This clearly changes the decision behaviour as it restricts the solution space. We believe this will be an important feature to resolve the time constraints specified for TASKA in D1.1.

Finally, we also found evidence of the importance of partitioning in scaling data processing applications. The worker configuration analysis in Figure 6.1.4 considers only a static partitioning technique that requires running beforehand and creates a copy of the whole dataset with the appropriate splits (data duplication). In the future, we will explore the inclusion of dynamic partitioning techniques, which avoid duplicating data in storage and enable faster partitioning times by creating virtual slices of data instead. This is done by generating navigable data indices and calculating data pointers. Implementing dynamic partitioning not only will allow faster partitioning times but also improve system scalability.

3. Data Mining

3.1. Overview

In this Section we discuss the workflow aspect of data processing in EXTRACT, with specific facets that pertain to Machine Learning (ML) support. As explained in the previous Section, datasets are first-class citizens in EXTRACT. An EXTRACT workflow may begin with some input datasets, and may generate datasets as part of its operation, in addition to other types of results (e.g., visualizations, notifications to external systems). Figure 3.1.1 below demonstrates the execution of an EXTRACT workflow as execution of one or more data processing steps. The actual orchestration of the workflow is discussed in D3.2.

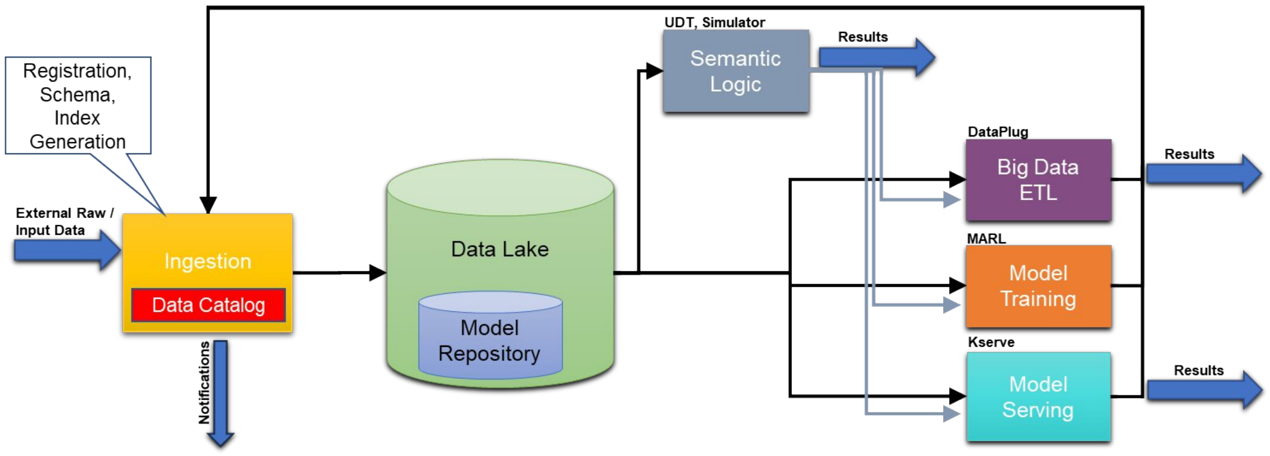

Figure 3.1 Data Mining Workflow Execution

As explained in Section 3 above, each dataset, whether created by a workflow or introduced from an external source, needs to undergo ingestion, which registers the dataset in the catalog of the data lake, and also builds and registers metadata descriptors, indices and other relevant details. Thus, ingestion as a workflow step may occur in the beginning of the worklow (e.g., for input data) and at every processing step that generates a new dataset, as explained further below.

Once the dataset is ingested, it becomes part of the EXTRACT data lake. This is true for any kind of dataset, and in particular, machine learning models. EXTRACT allows regarding models-as-data, which means encoding the model (inference algorithm) as a set of parameters, depending on the type of model being used. For example, a neural network can be encoded as a tensor (multi-dimensional array) of numerical weights, each pertaining to an edge between neurons. Models are stored in the model repository part of the data lake.

A workflow step may use one or more datasets from the data lake as input. Some of the data may be used as-is, whereas other data may require application-specific semantic representation, as is demonstrated in D1.2 for the PER use-case. This uses the semantic layer described in the previous Section about Data Infrastructure

The actual processing of a workflow step in EXTRACT may consist of different implementations depending on the specific application. It may involve generic ETL (Extract-Transform-Load) using the data staging facilities (see Section 3.3). This is a kind of processing that may consume datasets and creates new datasets. In EXTRACT, we use the same staging component, Lithops, to that end. Another option is to train a model, which is a new dataset and should therefore be ingested after being created. This is accomplished in EXTRACT using PyTorch. Last, it may serve a model, i.e., make it available for inference. This could be an online model service (as discussed further below). In EXTRACT, we use serverless Kserve for online serving. Alternatively, offline (batch) inference can also be done using PyTorch or Lithops and an input dataset, with the result dataset undergoing ingestion. Using this basic components can be later on expanded to complex workflows for online serving (e.g., combining multiple models).

3.2. Dataset Ingestion

The data ingestion process is essential for data management in a data lake, and as such, an important part of the EXTRACT architecture. It encompasses the acquisition, registration, and initial processing of data from the sources.

Overall, there are two main objectives in the ingestion process: metadata acquisition and data preprocessing. The specifics of these processes depend on the particular data formats being ingested and the procedures, workflows, or applications that typically process them.

In the case of metadata acquisition, information relative to data is provided along data insertion to the system, or extracted automatically from each dataset during the ingestion process. E.g., these metadata may include simple naming and tagging of datasets, location, and other characteristics of the conditions where it was generated and by who. These and also more complex properties enable the data to be identified and retrieved effectively from the data lake, and feed the basic features of the system to enable data discovery, localisation, and searchability. Further, this information may be required later by the data mining workflows. Interestingly, the data mining processes may require specific information or even the generation of indices that enable efficient navigation within the data set. These types of complementary information and indices are also generated during the ingestion process.

As for preprocessing procedures, some data formats and/or some data mining workflows require some data preparation or transformation that must be applied before the dataset is inserted into the data lake. Some examples of these transformations could be data filtering or compression that helps optimise storage utilisation and data retrieval. The nature of these preprocessing tasks varies according to the needs of the data type and format and aids the subsequent processing steps.

Ingestion may run anywhere in the continuum (edge to cloud) as detailed in D4.2. The ingestion processes are containerised to be deployed on the compute substrate in any continuum site and to be scaled dynamically to the income of data into the EXTRACT platform.

3.3. Model Repository

EXTRACT employs a model-as-data approach, where machine learning models are encoded as data files. Encoding is one of several options, such as algorithm arguments, e.g., the weights of a neural network, or as serialized code, e.g. pickled Python. There are several common formats for storing mode data, such as HDF5, JobLib, etc. In EXTRACT, we chose to start with Open Neural Network eXchange (ONNX - spelled "onyx"), which is one of the most popular formats, supported by many platforms (including PyTorch), and can be converted to and from many other formats.

Having the models encoded as highly-compatible datasets opens up several important capabilities: they become portable, can be delivered to different exploiters, e.g., for inference or tuning regardless of the recipient software. They can also be versioned to indicate model evolution, and moved closer to where they are needed.

When models are created in EXTRACT, they are stored in the model repository - a section of the data lake (typically, S3 bucket) that is set by the user's application to be used by both training and inference/serving. When a new model is created or updated (new version), it is stored in the respective model repository and ingested. Then, a recipient may subscribe to notifications on that repository and be notified by the catalog of the availability of the new model or version, and be able to access it.

3.4. Model Training

For the model development and training, we employ PyTorch, a leading deep learning framework known for its flexibility and efficiency. PyTorch provides us with flexible tools for building and training neural networks, allowing us to experiment with various architectures and optimization techniques.

Our model comprises two neural networks, each consisting of three linear layers. These networks are designed to extract features from the input data and make predictions based on the learned representations. The detailed explanation of the model architecture and algorithm can be found in D1.2. The number and type of layers, as well as hyperparameters such as learning rate, batch size, and hidden size, can be easily modified to fit the required use case.

The trained model is ready to be exported using the ONNX format, facilitating integration into various deep learning frameworks such as TensorFlow. This exportability allows developers to deploy the model across different environments and use cases.

Additionally, RAY is used to scale the training process seamlessly across multiple machines or GPUs. RAY allows to efficiently parallelise computations, significantly reducing training time while maintaining resource utilization.

3.5. Model Serving

EXTRACT model serving (model inference as a service) is implemented through Kserve, which is an open-source model serving solution. Kserve is in fact a "meta-model server", in the sense that it wraps concrete model servers in an elastic deployment that can scale out or in according to inference request demand.

The architecture of Kserve is shown in Figure 3.5.1 below. A first point to notice is that Kserve can serve models produced by all the major model training platforms in the markets (e.g., TensorFlow, PyTorch) and all the common formats (incl. ONNX). This is because Kserve embeds concrete model servers that are compatible with all the formats, such as Nvidia Triton MLserve, and many others.

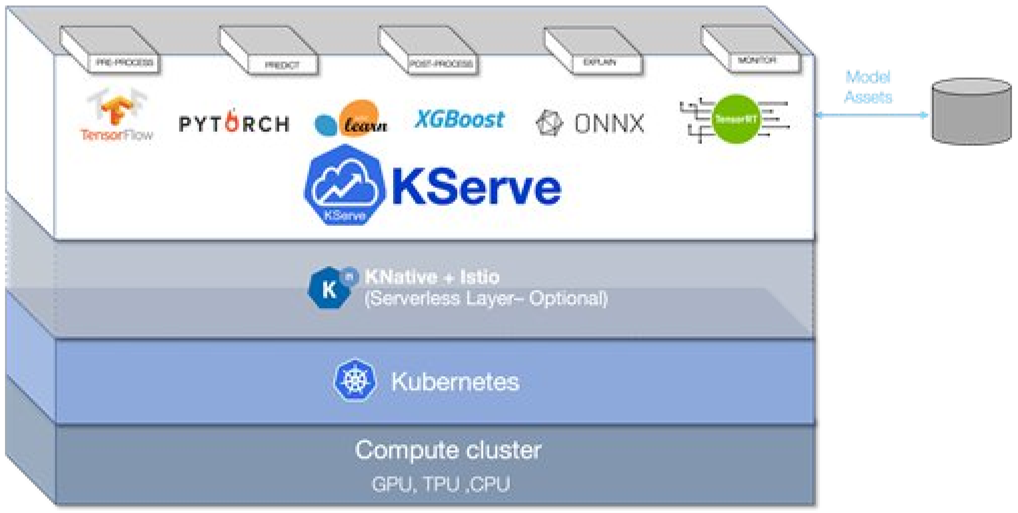

Figure 3.5.1 KServe Architecture

Second, note that Kserve typically uses Knative and Istio for a serverless deployment. Kserve predictors (models hosted on model servers) are deployed as Knative services, which allow scaling them dynamically out (increase the set of predictors) or in (decrease). The scaling is governed automatically by responding to changes in request load, detected by Istio. This allows Kserve to dynamically balance responsiveness and resource consumption / footprint.

Last, note that Kserve is capable of leveraging accelerators such as TPU and GPU for inference. This is important for reducing computation latency for some EXTRACT real-time applications, as described in D2.1.

Note that D2.1 suggested ModelMesh as a solution for efficient elastic large-scale model serving, whereas in this document we switch to serverless Kserve. Both solutions are large-scale, elastic and performant. However, they operate based on different principles - ModelMesh uses LRU cache for managing model elasticity, whereas Kserve uses serverless principles of response to demand. The switch was triggered by partner (IBM) interest change and by ModelMesh becoming less popular and attractive to developers compared to Kserve.

3.6. Semantic Logic

This is a logic level that transforms or augments the loaded data in the application domain prior to processing. This maps to the Semantic Layer introduced in the previous Section for Data Infrastructure. It can be an intermediate step in a workflow if required by the application.

4. Data Security, Privacy, and Integrity

4.1. Data Security

The general high-level Cybersecurity Architecture and Requirement of the EXTRACT platform are detailed in the deliverable D4.2, which encompasses data security aspects besides general cybersecurity, system security, software security and pipeline security considerations. However, we outline here certain cybersecurity requirements and architectural choices that are relevant specifically to Data Security as such, and these should be interpreted from the perspective of data processing along with some data privacy considerations. Nevertheless, data privacy deserves a separate consideration and therefore is outlined in a dedicated section focusing on specific data privacy state-of-the-art techniques.

- **Data Security (data-at-rest)**: EXTRACT architecture combines a high number of varied components (either local or distributed) that have communications needs in terms of data planes, control planes, and orchestration. Therefore, one of the main security requirements is to ensure that the communications occur in a uniform, interoperable and secure manner, and that the security and integrity of the data is also ensured. Also, it is important that the security of those communications is underpinned by state-of-the-art security parameters and configurations recommended by leading standardization bodies (e.g., NIST).
    - Whenever practically possible and applicable, all the data should be stored on encrypted volume files, whether those are native OS filesystems, ObjectStorage native encryption overlay, or encrypted docker volumes.
    - Data security for "RL Agent(s) (inference)" considerations
        - Any "model file" must be accompanied with at least its SHA256 (or SHA512) check value (usually in Linux stored as some-file-name.txt.sha256asc or some-file-name.txt.sha512asc)
            - The type of hash, SHA256 or SHA512, MUST NOT be hardcoded and MUST BE a configuration value to allow easier change, easier testing and security-future-proof upgrades
        - Every time when loading any "model file"
            - According to hash type, check if corresponding sha256asc / sha512asc file exists for the given model
            - If sha256asc / sha512asc does not exist, fails to load (empty or bad permissions), or the SHA256/SHA512 value of the model files differs from the value supplied inside its corresponding sha256asc / sha512asc file ⇣ DO NOT LOAD, display error/warning, continue/exit gracefully (depending on the criticality of failed-to-load file and the design of the module)
            - NOTE: Because reading/loading large files and computing hash value of large files has performance/time/resource cost, it is advisable to design "read-and-hash-and-checkhash" routines (should be possible in most modern programming languages) that would read the bytestream of a file and compute hash at the same time
    - General Data Integrity considerations

- Data Integrity is one critical component and requirement for an overall trustworthy computing environment - in the end, if data integrity cannot be guaranteed/verified or at least data tampering detected, how can a system be trustworthy?
- For MVP, the general concept is to use non-weak, state-of-the-art, industry standard hashing algorithms, such as SHA256 with preference and compatibility towards SHA-3 families such as Blake3. The current industry standard is to use SHA256 and this is a baseline that is also used for the EXTRACT platform cybersecurity architecture requirement. Nevertheless, we aim for Phase3 and Phase4 consideration and evaluation (especially performance impact) for using Blake3 in certain, or all, parts of the system (depending on their ratio of computational power vs. their data integrity realistic risks).
- For PER use-case, it was concluded that data integrity is important in all aspects and data flows, but most importantly in the "instructions to users" and "users location and meta-data" set of data fields, therefore API-level data exchanges (e.g., JSON) are to be additionally digitally signed at the "application layer" (in ISO OSI 7 layers terminology) using for example ECDSA.
- For TASKA use-case, it was concluded so far that data integrity is generally important (but not absolutely important as in "life-and-death" scenarios of PER use-case), while data privacy is not considered a risk nor a strong requirement. However, given the extreme volumes of data, it is impractical (and most likely infeasible) to hash and/or digitally sign ALL the TASKA Raw Data, therefore only certain parts and fields of the post-processed data will be subject to strong hashing and digital signatures, in addition to TASKA-specific non-corruption checks of filesystems/directories/files using (and improving) TASKA/OBS internal tools that exist already.

4.2. Data Privacy

Data Privacy ensures the privacy of individuals by protecting their sensitive data. Privacy can be achieved by combining different implementations and avoiding collecting unnecessary data so it can be divided into privacy by design or by default.

- Privacy by Design

1. Communication only under https protocol
2. Anonymization/pseudo-anonymization of data
3. State-of-art privacy implementation (GDPR Compliant)

- Privacy by Default

1. Minimize the amount of data harvested
2. Minimize the Time to live of the personal information

EXTRACT intends to follow best practices to ensure data privacy when arriving into production where real users data shall be used (mainly in the PER use case). However, EXTRACT will also contribute to the research and implementation of state-of-art privacy preserving techniques before using the real users data. *Differential Privacy* is hugely used to ensure the privacy of individuals in any datasets. The main idea is to give a modified version of the data that can preserve the general information of the original data but with no link to any individual record or user. Which data is considered sensitive and needs to be protected will be clearer through the life of the project and need to be discussed with the Ethical Manager of EXTRACT, however, the first scan of the existing methods and techniques shows that users location might be sensitive information depending on the use of the location. This need allowed to envision two different scenarios as follows:

- First scenario:

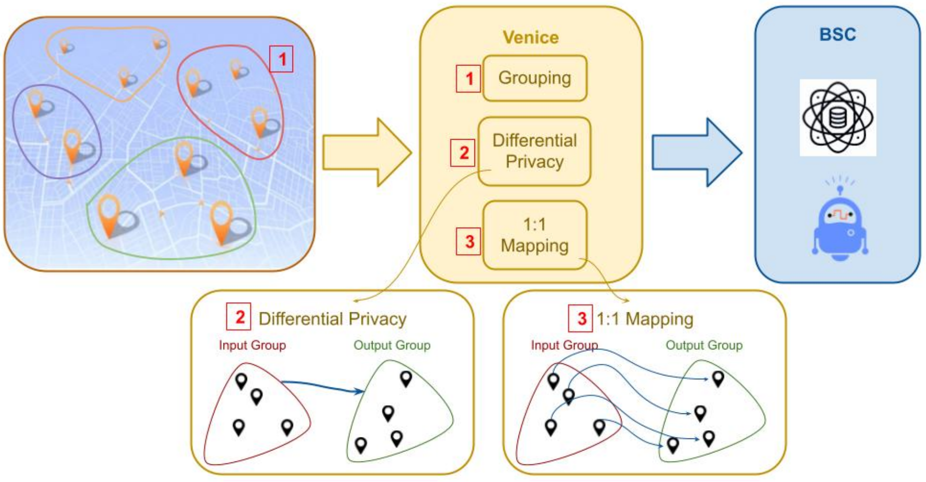

Figure 4.2.1: the first scenario of Differential Privacy

Figure 4.2.1 shows the first scenario where the involved parties in the PER use case are concerned with privacy preservation, which are the mobile phones of the users, Venice city and BSC as the model provider to be using the users data. The main steps of the scenario are:

1. Grouping of the users geographically
2. Differential Privacy by sending the same number of users in each group but with slightly different locations (by adding random noise)
3. The mapping between the original locations and the Differential Privacy locations to send back the reply to the users

- Second scenario:

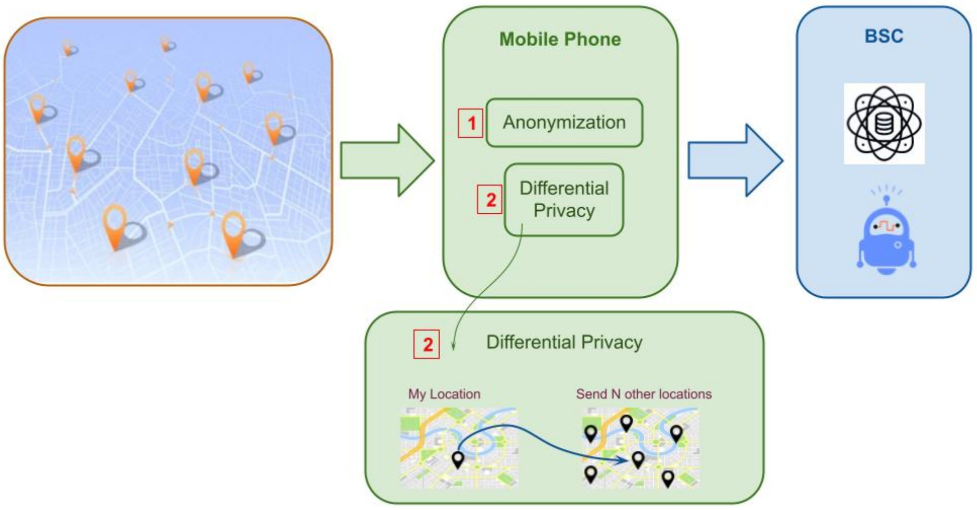

Figure 4.2.2: the second scenario of Differential Privacy

Figure 4.2.2 shows the second scenario where the involved parties in the PER use case are only the mobile phones of the users and BSC. The main steps of the scenario are:

1. Anonymization of the user/phone identity
2. Differential Privacy by sending the location of the phone with N other locations generated randomly near the original location

Comparing the two scenarios is presented in Table 4.2.1.

Scenario 1	Scenario 2
One request by group	N+1 requests by user
A server is needed in (or owned by) Venice	No server is needed in Venice
Position info is not necessarily private to Venice	Position info is also private to Venice

Table 4.2.1 comparison between the two proposed scenarios

4.3. Model and Computation Protection

The Multiparty Computation (MPC) is used mainly for data/model security, to ensure the data and the model weights to be secure during computation while protecting the model against counter examples. This computation is used when more than one party or entity wants to perform a common task or computation without sharing their assets (data or model), where they all have the final result decrypted. These parties can provide data, model or function or just computation resources.

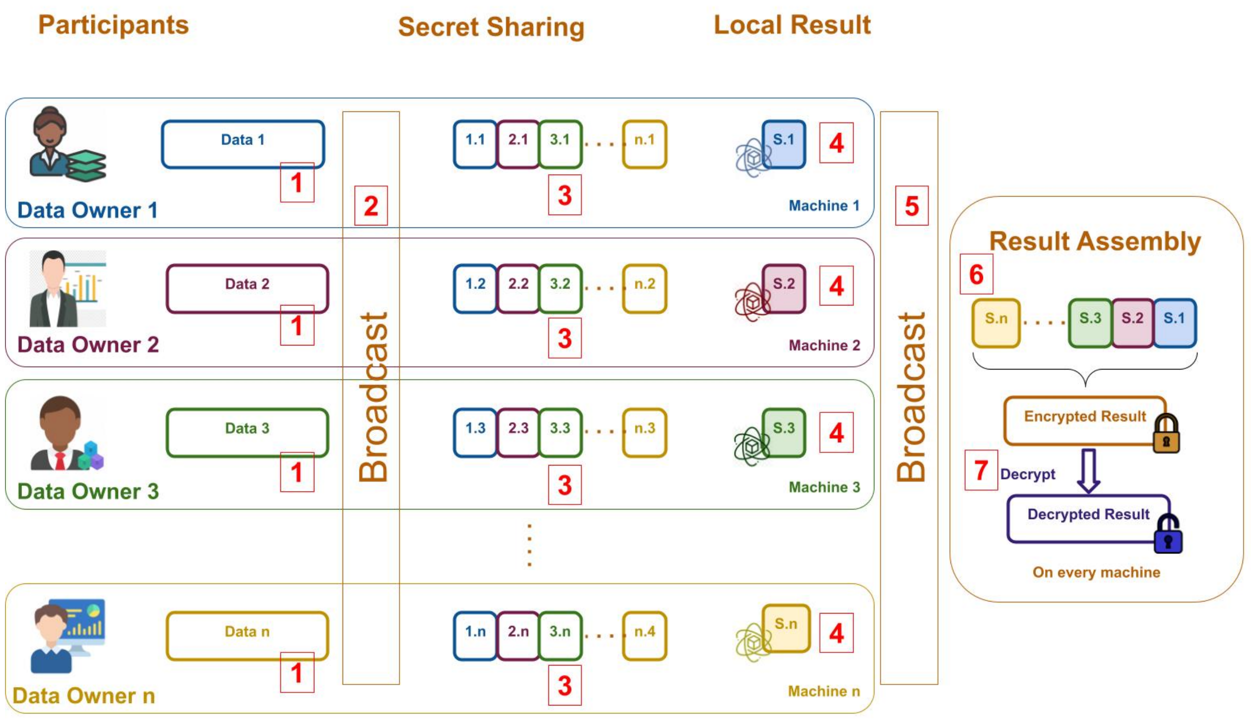

Figure 4.3.1: the general view of Multiparty Computation (MPC)

Figure 4.3.1. Shows the steps of the MPC protocol, assuming that every party involved in the computation will participate with a machine that they own totally (each machine is owned only by its owner). The main steps of the protocol are as follows:

1. Data/Model owner has a dataset or a model to participate in the computation, these datasets will be partitioned randomly into shares
2. Each party will send the shares to the other parties, as the shares are random, the other parties can not get any information about the original data just from the received share
3. Each party has a share from each other party
4. Each party locally computes the results of the received shares and model
5. Each party broadcasts the result to the other parties
6. Each party can reconstruct the whole encrypted result from all received parts
7. Each party can decrypt the encrypted result to have the meaningful result

5. First MVP

5.1. Restrictions & Relaxations

The objective of the first MVP is to demonstrate basic capabilities of the EXTRACT platform. In that sense, we do not expect to see full end-to-end component integration but rather provide a sense of what the eventual system would be with independent illustrations of specific components. In coordination with D4.2, the MVP for WP2 includes a collection of core components (catalog, Lithops, model serving - Kserve, model training) in accordance with the agreed delivery level for this deliverable (R - report). Beyond the agreed level, we provide demonstrators based on voluntary will of partners.

5.2. Demonstrator – ONNX Model Serving on Kubernetes using Serverless KServe

As already mentioned in Section 3.5 we use serverless Kserve for online serving in EXTRACT. We point the reader to Section 3.5 for an overview of Kserve.

5.2.1. Demo Story

The demonstration is about showing how the Kserve stack can be installed and how inferences can be sent for a model in ONNX format. More specifically we will demonstrate, after setting the InferenceService machinery how inferences can be sent through a Jupyter notebook.

In the demo setup, we show in a linux environment (Fedora 39) how to create a single node kind cluster and how to install Kserve on this cluster.

We then show how to create the InferenceService that will handle the inference requests sent, in our case, through the Jupyter notebook.

5.2.2. Demo Execution

Environment. Prepare a Linux environment which can be a bare metal machine or a VM. This environment assumes that kubectl has been installed (we used latest version v1.29.2) and that its current context is a Kubernetes cluster (a single node as can be obtained with kind is possible). The Kubernetes server version that we used was the version of the kubectl client.

Setup. you need to prepare:

1. **kubectl tool**
   We followed the instructions detailed at Install and Set Up kubectl on Linux | Kubernetes, however kubectl can be installed by using package management as detailed in Install and Set Up kubectl on Linux | Kubernetes
2. **Python tooling with Python virtual environment**. We recommend to install pyenv as a way to do this. Then, we create a Python 3.12.2 virtual env called onnx_venv using the following commands:
   ```
   pyenv install 3.12.2
   pyenv virtualenv 3.12.2 onnx_venv
   pyenv activate skystore-test
   ```

Execution:

1. This step may be skipped if you already have access to a Kubernetes cluster.
   Following instructions permit to create a single node cluster with kind:
   a. `export CLUSTER_NAME="km2"`
   b. Create kind cluster:
      ```
      cat <<EOF | kind create cluster --name ${CLUSTER_NAME}
      --image "kindest/node:v1.29.2" --config -
      kind: Cluster
      apiVersion: kind.x-k8s.io/v1alpha4
      nodes:
      EOF
      ```
2. Install the Kserve stack as detailed in https://kserve.github.io/website/0.11/get_started/
   Note that we will use latest version of Kserve : 0.12 (it has no documentation for now)
   a.
      ```
      curl -s
      "https://raw.githubusercontent.com/kserve/kserve/releas
      e-0.12/hack/quick_install.sh" >
      quick_install_kserve_v0.12.sh
      ```
   b. `chmod 755 quick_install_kserve_v0.12.sh`
   c. `./ quick_install_kserve_v0.12.sh`
      ```
      In the quite verbose output, you should see the
      following lines:
      Successfully installed Istio
      Successfully installed Knative
      Successfully installed Cert Manager
      Successfully installed Kserve
      ```
3. Check that the out-of-the-box inference runtimes are installed:
   ```
   kubectl get clusterservingruntimes -A
   ```

You should have 10 ClusterServingRuntimes where the one which we will use in this demo is kserve-tritonserver since it the the only one which can handle models in ONNX format.

4. Create a namespace (e.g., kserve-test) for our tests:
   a. `kubectl create namespace kserve-test`
   b. `kubectl config set-context --current --namespace=kserve-test`
5. Create the InferenceService
   a. Create file onnx-gcloud.yaml which should contain following text (see also the "new schema" at https://github.com/kserve/website/tree/main/docs/modelserving/v1beta1/onnx)
   ```
   apiVersion: "serving.kserve.io/v1beta1"
   kind: "InferenceService"
   metadata:
     name: "style-sample"
    spec:
    predictor:
       model:
         protocolVersion: v2
         modelFormat:
           name: onnx
         storageUri: "gs://kfserving-examples/models/onnx"
   ```
   b. `ku apply -f  ./onnx-gcloud.yaml`
   c. Check InferenceService status with command
   ```
   kubectl get InferenceService style-sample -n kserve-test
   ```
   The status of the created InferenceService `style-sample` until its status becomes ready.
   Note that it may take some time (depending on network bandwidth it may reach 10 minutes)
   d. Move to the onnx_venv virtual environment (see second bullet of the setup)
   ```
   source ~/onnx_venv/bin/activate
   ```
   e. Install the libraries that will be needed to run the Jupyter notebook:
   ```
   pip install -r   ./requirements.txt
   ```
   Where `cat  ./requirements.txt` yields:
   ```
   jupyter
   numpy
   pillow
   protobuf
   requests
   ```
   f. We have to find out the name of the istio ingress gateway:
   ```
   kubectl get svc --namespace istio-system --selector="app=istio-ingressgateway"; echo ""
   ```
   which should output something like:
   ```
   NAME   TYPE  CLUSTER-IP  EXTERNAL-IP   PORT(S)  AGE
   ```

```
istio-ingressgateway   LoadBalancer   10.96.164.164
<pending>
15021:31035/TCP,80:30468/TCP,443:31473/TCP   4h21m
```

g. We detect the Istio ingress gateway:
   `INGRESS_GATEWAY_SERVICE=$(kubectl get svc --namespace istio-system --selector="app=istio-ingressgateway" --output jsonpath='{.items[0].metadata.name}')`
h. So that command: `echo $INGRESS_GATEWAY_SERVICE` yields `istio-ingressgateway`
i. We now port-forward local port 8080 to port 80 of the ingress gateway service: `kubectl port-forward --namespace istio-system svc/${INGRESS_GATEWAY_SERVICE} 8080:80 &`
j. Before invoking the Jupyter notebook:
   i. prepare a few (royalty free) images
   ii. download from https://github.com/kserve/website/blob/main/docs/modelserving/v1beta1/onnx/mosaic-onnx.ipynb the `mosaic-onnx notebook`
k. We invoke the Jupyter notebook with command: `jupyter notebook`
l. Open the mosaic-onnx notebook
m. Within the notebook you should replace the `image.jpg` by one of your images
n. The final cell of the notebook should show a segmented image

5.2.3. Demo Video

The video linked below delivers the demo story as explained in 5.2.1 and 5.2.2. To avoid a lengthy video, we skip the whole setup part and present just the demo itself, of creating the InferenceService and issuing inferences through the notebook.

https://b2drop.bsc.es/index.php/s/bW34Aqcc97tqT5e

5.2.4. Demo Resources

Kserve deployment repository at GitHub:
https://github.com/ymoatti/kserve_deployment

5.2.5. Next Steps

This demo presented a basic capability of serving models. Kserve can practically serve all models in the market, but we focus on ONNX, which is the agreed model format for EXTRACT. There are quite a few improvements that we intend to implement going forward with the project, including:

- Support for reading models from local S3 - Minio/Ceph
- Separating the inference from requests - i.e., not invoke inference on every request but return a cached result until a separate trigger re-invokes inference
- Performance/utilization improvements (still TBD)

6. Evaluation

6.1. TASKA-C scalable computation using Lithops

To present and evaluate the need for a smart-provisioning layer in Lithops and show its potential, this section explores the idea in the context of TASKA C (described in D1.2). We will show the problems that arise in the execution of this extreme data workflow and the benefits of adding the ability of automatically choosing the right scale to perform its different tasks in each execution.

Roadmap

We start by evaluating the different tasks that are part of TASKA C. As described in D1.2, TASKA C workflows are compositions of several data processing, each presenting an opportunity for optimising its scale. We will see that the Rebinning task is specially demanding due to being the most data-intensive one. Therefore, the next parts of this exploration put special focus on it.

Then we continue by comparing different scale-out strategies, employing cost and execution time minimization as our objective function. We employ a pareto frontier analysis to identify optimal configurations by balancing the trade-offs between execution speed and cost.

Finally, we refine our analysis by incorporating the partitioning time in the scale out approach and comparing it against a scale up strategy, this offers a more nuanced understanding of the scalability challenges and solutions in distributed computing environments. This analysis ends in a set of conclusions that validate the benefits of a smart-provisioning layer in Lithops.

Use Case Overview

Computational Requirements and Throughput Analysis

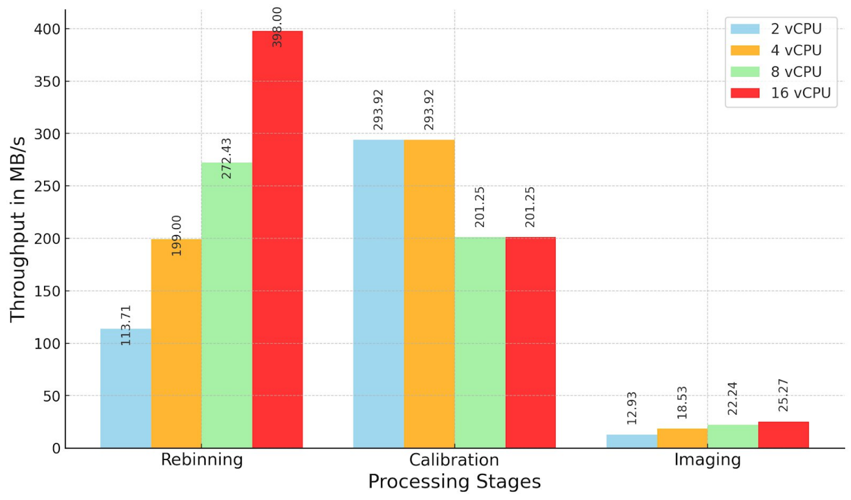

Figure 6.1.1: Throughput per step with different VCPU configurations.

We start with the analysis of how the throughput of the current implementations of each computational step (rebinning, calibration, and imaging) answers to varying numbers of CPU cores (vCPUs in the cloud). The results are presented in Figure 6.1.1.

The Rebinning step exhibits an increase of throughput with additional vCPUs, which indicates a benefit from scaling up resources. However this trend does not hold for the Calibration step. We observe that increasing the vCPU count does not yield higher throughput, suggesting that the calibration step does not scale with additional computational resources, highlighting a limit on scalability. We also observe that the calibration step throughput decreases with more vCPUs provided. This is due to the apply calibration substep within and we will explore this issue further in the future. Finally, Imaging does scale up, although very slightly. This suggests that the Imaging process, although using parallelism, requires significant synchronisation that hinders its scalability.

Given these findings, we conclude that Rebinning and Imaging benefit from a scale up approach, while Calibration does not.

We see that each process (step) has its own particularities and their computational requirements vary. For instance, we can observe that by providing more vCPUs to the Rebinning step as well as the Imaging step it is possible to improve the throughput, but at very different rates. Further, in the Calibration step, a scaling up approach does not yield performance improvements, and thus a different strategy must be used to correctly scale its computation. This clearly shows that provisioning the right amount of resources for each of these steps is not general and requires a smart decision that takes into account the particular characteristics of each task.

Data Volume Analysis

Once we understand the performance of the different steps, we also want to understand how they interact with data. For that, we next study the pattern on data usage as well as ratios in input and output data for each step to identify what processes are the most computational/data expensive and ultimately identify how data volume affects their execution. To this end, Figure 6.1.2 plots the input and output volume of each step for processing an input dataset of close to 8 GB in a simple sequence of rebinning, calibration, and imaging.

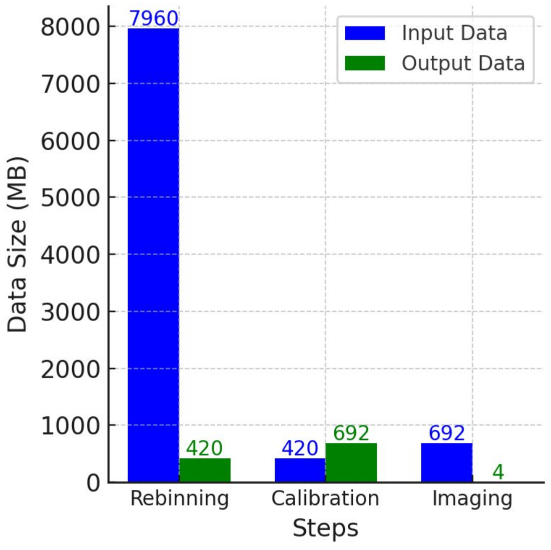

Figure 6.1.2: Data volumes per step (input-output).

First, rebinning is the step that ingests most data at 7960 MB (the full dataset). This is a lossy process that compresses the data, which in this case reduces the output of the process to 420 MB.

The calibration step takes a measurement set (usually after rebinning) and adds a new column to it. Consequently, the calibrated measurement set has a higher size compared to the original measurement set.

The imaging process takes the calibrated data generated in a calibration step and creates an image cube with a human-readable representation of the instrument observation. In this case, the resulting image is 4MB in size.

We see that the steps have clearly diverse data requirements, with some of them processing large data with lightweight computation, while others do not process much data and are heavier on the logic. Again, this requires an informed solution that cannot be general and strengthens the idea of a staging layer that smartly chooses the right strategy for us not only depending on the computational needs but also on the data processed of that step; i.e., a data-driven decision.

Finding the right size and number of workers

The Scale-Out vs. Scale-Up tradeoff

After the analysis of the different steps in TASKA C, we focus on the question of finding the appropriate worker configuration for them. To wit, we want to know what is the best combination of scale-up (growing the workers with more resources) and scale-out (adding more workers). For this, we analyse the rebinning step in more detail. First, we explore data and problem partitioning for the rebinning step to understand how it behaves when scaling out. Then we compare it against a full scale up approach, where the data is ingested whole on a single machine with multiple vCPUs. The objective of these experiments is to give an empirical overview of the different options available to the smart data-staging layer, what it will need to take decisions on, and why it is needed in the case of workflows that deal with extreme data.

To find the best configuration of workers, we consider two variables that represent our objective: execution time and cost. Time is a direct representation of application performance, while cost is also important if we want to ensure efficient execution without overwhelming our resources or wasting unnecessary energy. We want to optimise both cost and time since the objective is to have executions that take the least time possible at the lowest cost possible.

Cost can be calculated simply as resource occupancy for a period of time, but to be precise, here we calculate it as the product of execution time (in ms) and the amount of resources utilised (in MB). To simplify, we use a system akin to resource billing in AWS Lambda, where the price is established for an amount of memory but represents a slice of a machine including proportional memory, CPU, and network.

In this experiment, we use several worker configurations (with different memory and proportional CPU) and plot them on the axis of execution time and cost. The first result is shown in Figure 6.1.3, where we explore how much it takes to process a total of 1091 MB of data with configurations ranging from 1 to 9 workers. For that, a 1090 MB single MS dataset is taken and partitioned into different chunk sizes. Then the rebinning step is executed for the various configurations. Each configuration has been run for 5 to 10 repetitions and the point represents the mean of the execution time and cost, with bars showing the error in execution time (horizontal) and cost (vertical).

To find the best configurations, we may first obtain the Pareto frontier. It will tell us the Pareto-optimal configurations for both variables. An optimal configuration is achieved when it is impossible to improve the performance or cost-effectiveness for one aspect (cost or execution time) without negatively impacting the other. In other words, a configuration is considered optimal if any attempt to reduce costs or decrease execution time leads to a detriment in the other area.

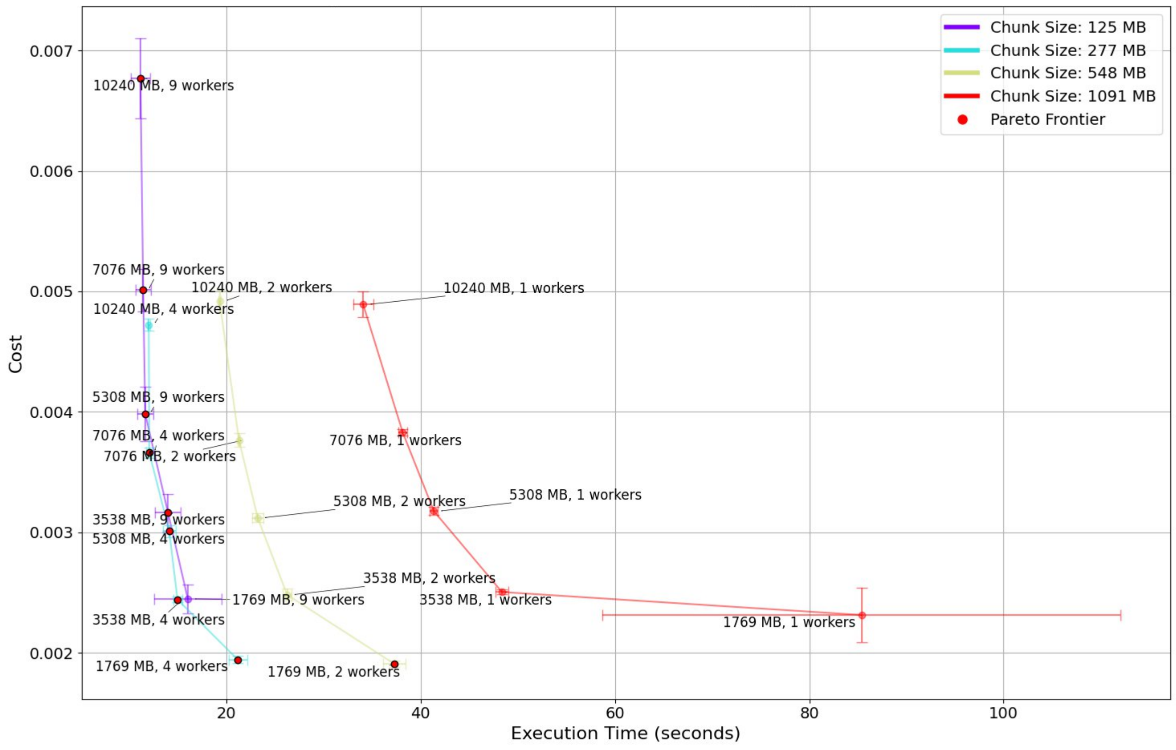

Figure 6.1.3: Pareto frontier for averaged executions of rebinning step with 1090 MB.

The Pareto frontier shows that the most optimal configurations are those that split the input data in small chunks (125 MB or 277 MB) and use multiple (small) workers to run the process.

This would allow the data-staging Lithops component to make decisions at runtime based on previous executions, if the scientist needed a fast execution, the Lithops data-staging component would make the workers as big as possible and have as many as possible (10240 MB of runtime memory, 125 MB of chunk size and 9 workers). On the other hand, if there is no execution deadline, the data-staging component would choose an equilibrated configuration with lower cost (a potential heuristic could be the point that minimises the distance to the origin, for example, and we would be considering using partitions of around 277 MB and workers of 3 GB of memory).

Incorporating Data Partitioning Time

The previous experiments do not reflect the time it takes to partition the measurement set of 1090 MB into smaller chunks. Data partitioning is an expensive process, especially for extreme data, required to enable problem distribution onto multiple workers. This process must run before the actual task (e.g. rebinning) to generate data slices from an original file that each worker will take individually. To build a fair comparison, we must add the partitioning time to each configuration that requires it.

For that, we have taken the casacore library and partitioned the measurement set of 1090 MB into a different number of partitions, trying also to scale the resources where partitioning takes place (bigger VMs with higher core count). With this evaluation, we find that creating these slices is a sequential process that does not scale with multiple CPU cores (i.e., the process takes a similar time with 2, 4, or 8 cores). Also, the process is mostly independent from the number of slices created, with partitioning time varying only up to 11% from creating 2 to 10 partitions. As a context from our running evaluation, partitioning time for an input file of around 1 GB takes between 46 to 52 seconds to complete. Of course, this process has its associated cost, which we must consider for configurations that require it (those with multiple workers).

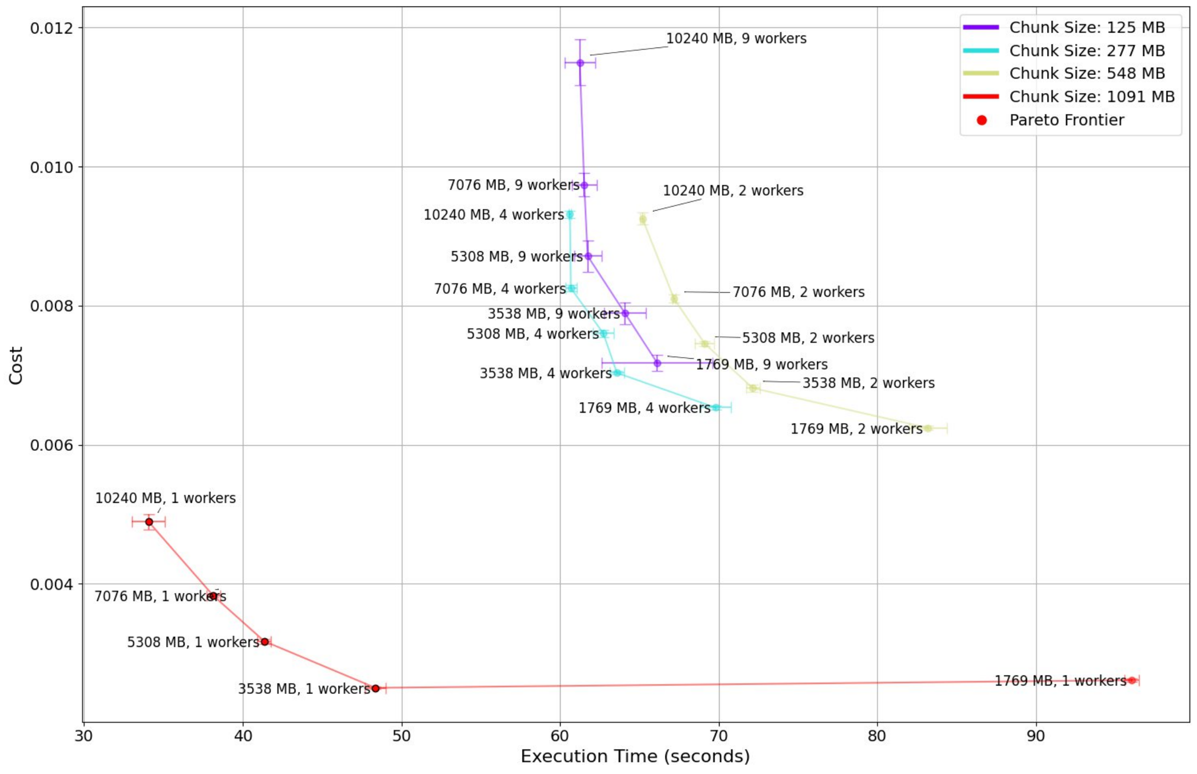

Figure 6.1.4: Scale up vs scale out for 1090 MB rebinning.

Figure 6.1.4 adds the time and cost of partitioning to the executions of Figure 4. This clearly creates a contrast between the configurations that use an exclusively scale-up approach (1 worker) against those that combine both scale-up and out (requiring multiple workers and, thus, partitioning).

When accounting the partitioning time for scaling out, we can observe that the cost of partitioning clearly exceeds the performance benefits of using multiple workers. Hence, using a single worker is much more efficient in both execution time and cost, unless the data does not fit into any available worker, where we would be forced to partition it.

We draw the following conclusion for the rebinning process of TASKA C: given a measurement set, it will be always better to ingest it as a whole instead of partitioning it. Therefore, this is the decision to be expected from a smart data-driven provisioning in Lithops. This is because rebinning scales vertically effectively, and utilises the all cores available in the worker. However, we still must consider a limitation: if the measurement set were big enough that it does not fit into a single worker, then we would still need to do partitioning. This fact also gives more strength to the Lithops data-driven smart provisioning layer, as it dynamically adjusts resources based on the workload and computational requirements. By intelligently managing the compute resources, it ensures that the data processing is both efficient and cost effective.

Conclusions

The experiments illustrate that there is no silver-bullet when scaling to face the diverse computational demands presented by the different workflow tasks. This variability demonstrates a critical need for a provisioning layer capable of dynamically selecting the most appropriate strategy in real-time, customised to the computational as well as data demands of each task. On one hand, some tasks may require more or fewer computational resources for the same volume of data based on the characteristics of its logic (compute or data-intensive). On the other hand, there is data variability, where each task execution may have huge variances in its data demands and require adaptive algorithms that efficiently manage the vast ranges of data sizes. This shows the need of an orchestration layer that can intelligently adjust to both computational demands and extreme data.

In our exploration, we have also seen that manual configuration and decision-making are time-consuming and also prone to inefficiencies. The smart provisioning layer envisioned in the Extract project will eliminate these challenges by automating the decision-making process, leveraging data properties, historical execution information, and the current resource availability to make informed provisioning choices that enhance both performance and cost-effectiveness of applications.

6.2. Model Protection Using MPC

The MPC, as a model security solution, provides the security of the Machine Learning models, and applications, against the adversarial threats when applied on the data either in training or inference phases. However, implementing MPC is not cheap regarding performance and accuracy requirement, in this section we present and evaluate the MPC in a multi machine scenario. The scenario was implemented including three parties to simulate the future potential collaboration in the PER use case between the parties VEN (data provider), BSC (model provider) and Cloud Provider for computation of the inference of a machine learning model on a validation dataset in a secure way that guarantee the security of the model and the data. As shown in Figure 6.2.1, the parties are renamed as Alice, Bob and Claude respectively. The assumptions of this scenario are:

- The scenario have three different parties involved in the inference computation
- Each partner participates with a machine/cluster in the MPC protocol separated from the others
- Alice has sensitive data and it is necessary to protect it
- Bob's model needs protection

Three docker containers were used to simulate 3 parties' machines, and this proof of concept used the MNIST dataset with a 2-convolutional layered model.
MNIST dataset. The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is publicly available and commonly used for training various machine learning models. The dataset size is

21.00 MiB, The number of samples (28×28 pixels images) divided into a training dataset of 60,000 images and testing dataset of 10,000 images.
Used hardware and software. The scenario was conducted using virtual machine running Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-164-generic x86_64) with 4 core Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz and 12 GB RAM

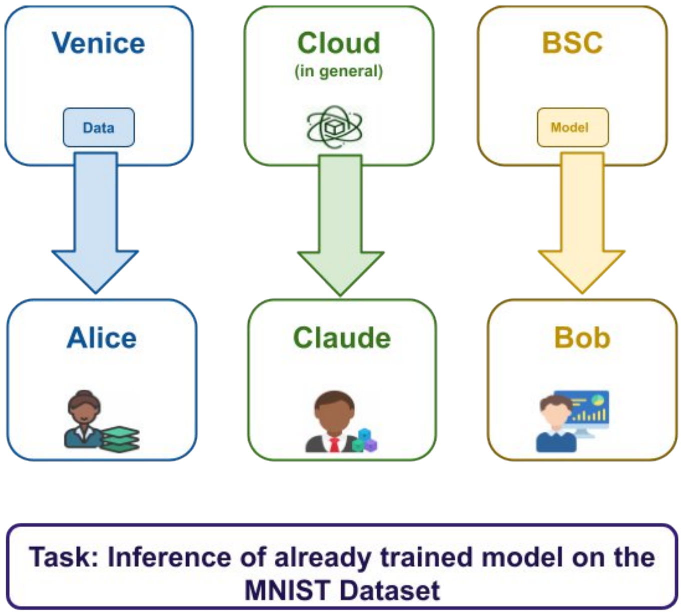
Venice
Data
Cloud
(in general)
BSC
Model
Alice
Claude
Bob
Task: Inference of already trained model on the MNIST Dataset

Figure 6.2.1: the simulation scenario and the mapping with the PER use case

The MPC library (CrypTen):
To implement the MPC protocol we used CrypTen, an open source library (GitHub link) for Privacy Preserving Machine Learning built on PyTorch. Its goal is to make secure computing techniques accessible to Machine Learning practitioners. It currently implements Secure Multiparty Computation as its secure computing backend.

MVP overview:
The scenario overview as shown in Figure6.2.2 involves three separated machines (Alice, Bob and Claude), Alice has the MNIST validation dataset with the correct labels while Bob has the trained model on the MNIST training dataset. The scenario steps are as follows:

1. All the three machines initialize the protocol and exchange the setup information
2. Alice encrypt the validation dataset and broadcast it to the others (send the secret shares)
3. Alice encrypt the trained model and broadcast it to the others (send the secret shares)
4. Every party calculates the inference of their model and data shares locally
5. Every party broadcast their result to the others and now everyone can reassemble the encrypted result
6. Every party can decrypt the result to have the predicted labels
7. Alice benchmark with the real label as the owner of the true labels

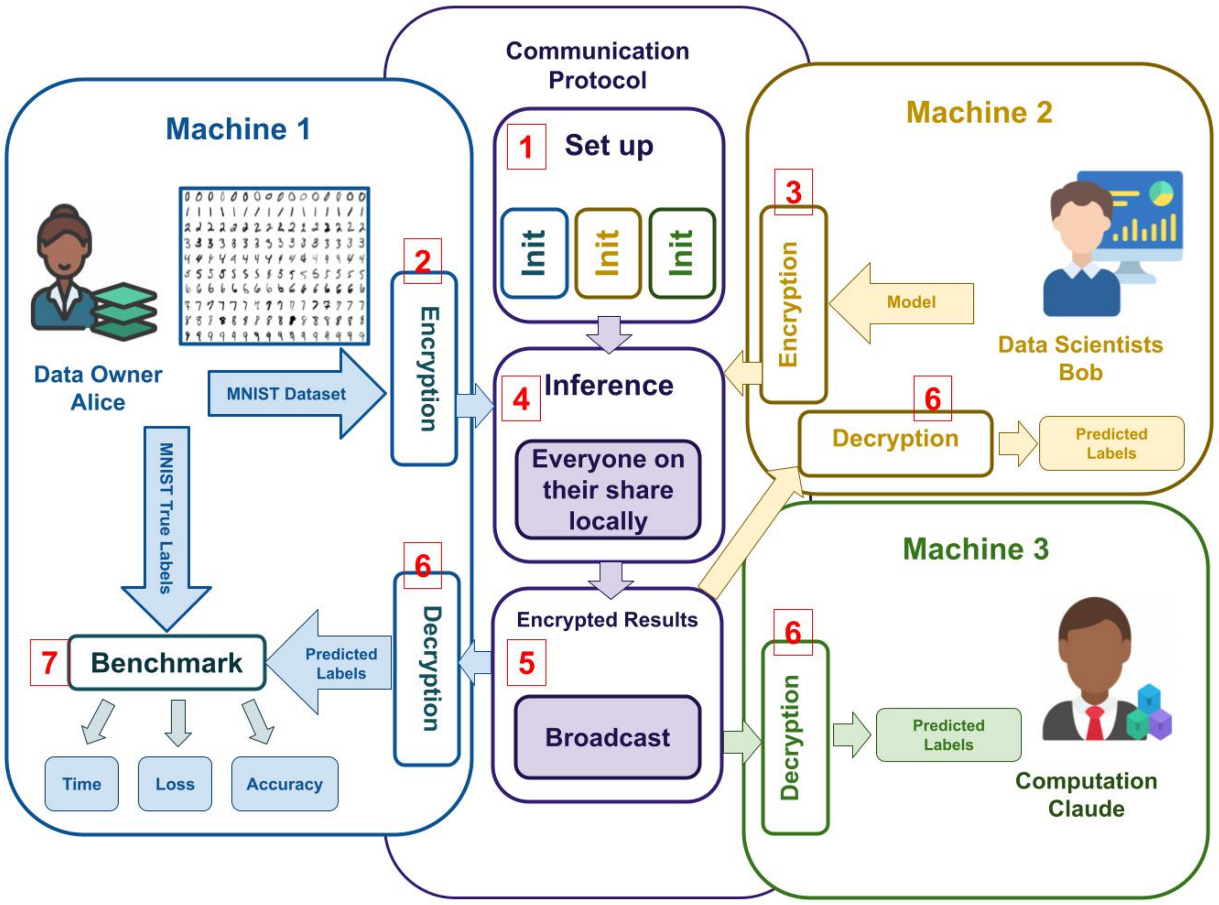

Figure 6.2.2: the MVP overview

Version 0:
The first benchmark to test the library on one machine with different models' structures and two multiprocess (Alice and Bob) on one machine, the goal was to test the functionality of the library and to benchmark the time and accuracy of using it.

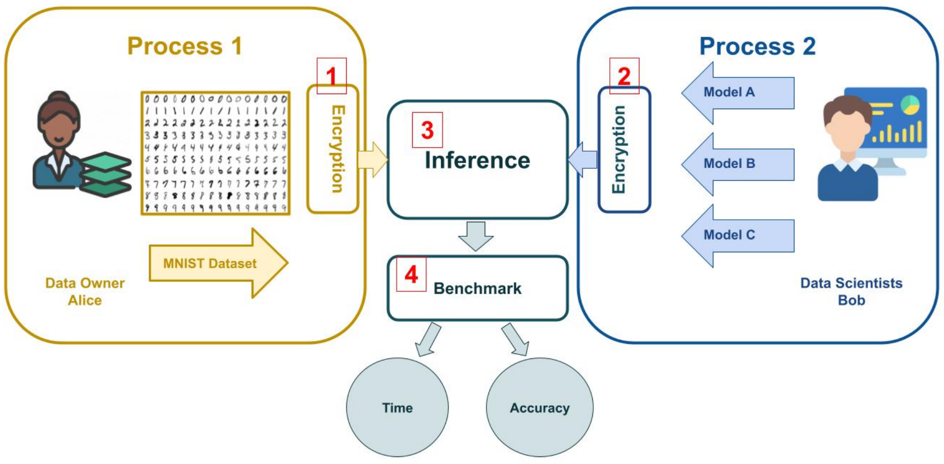

Figure 6.2.3: the MVP version 0

Alice has the MNIST validation dataset with the correct labels while Bob has the trained model on the MNIST training dataset. The scenario steps are as follows:

1. Alice encrypt the validation dataset and broadcast it to Bob (send the secret shares)

2. Alice encrypt the trained model and broadcast it to Alice (send the secret shares)
3. Every party calculates the inference of their model and data shares locally
4. Alice benchmark with the real label as the owner of the true labels

Bob has three model structures models A,B and C shown in Figure 6.2.4, where model A is a Multilayer Perceptron (MLP) structure with 3 linear layers, while models B and C are 2-convolutional layered models with different settings.

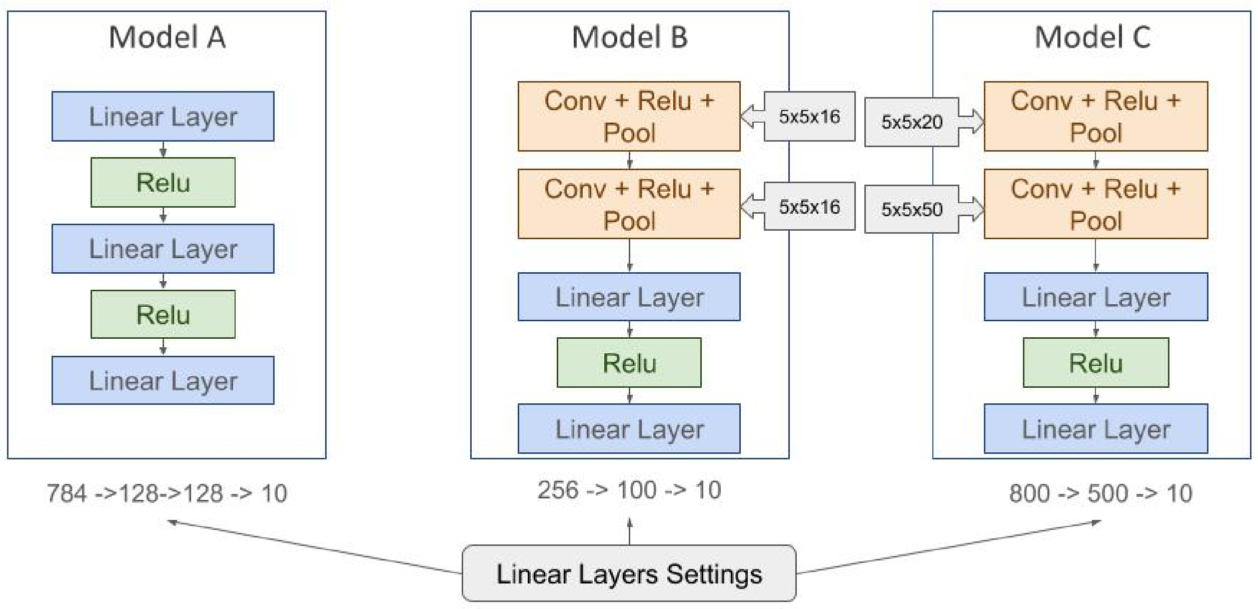

Figure 6.2.4: the different model structures A,B and C

The benchmark of this version is shown in Table 6.2.1, which shows the average time (in seconds) of the three models inference on 40 of validation batches

	Model A		Model B		Model C	
	Pytorch	With Crypten	Pytorch	With Crypten	Pytorch	With Crypten
Time	0.055	1.854 (33.7 times)	0.159	17.465 (110 times)	0.129	22.891 (177 times)

Table 6.2.1: the time benchmark of the model structures A,B and C compared with and without using Crypten

Table 6.2.1 shows the computation/performance of using the MPC (implemented by Crypten) meanwhile the accuracy was intact by using Crypten and for the next step, Model B was selected only.

MPC full MVP on three different machines:

To simulate three separated machines, docker was used to build a container and the compose file to set up the three different machines and the network that they communicate through. Figure 6.2.5 shows the three logical layers, 1) the file system with the validation dataset and labels, the trained model file and the script that the three containers will run. 2) the building of the docker python container with all the dependencies and the inference script file to be run. 3) run three different instances Alice (with access to the validation dataset and labels files), Bob (with access to the trained model file) and Claude. Also the compose file is configured to set up a local network to allow the three machines to communicate.

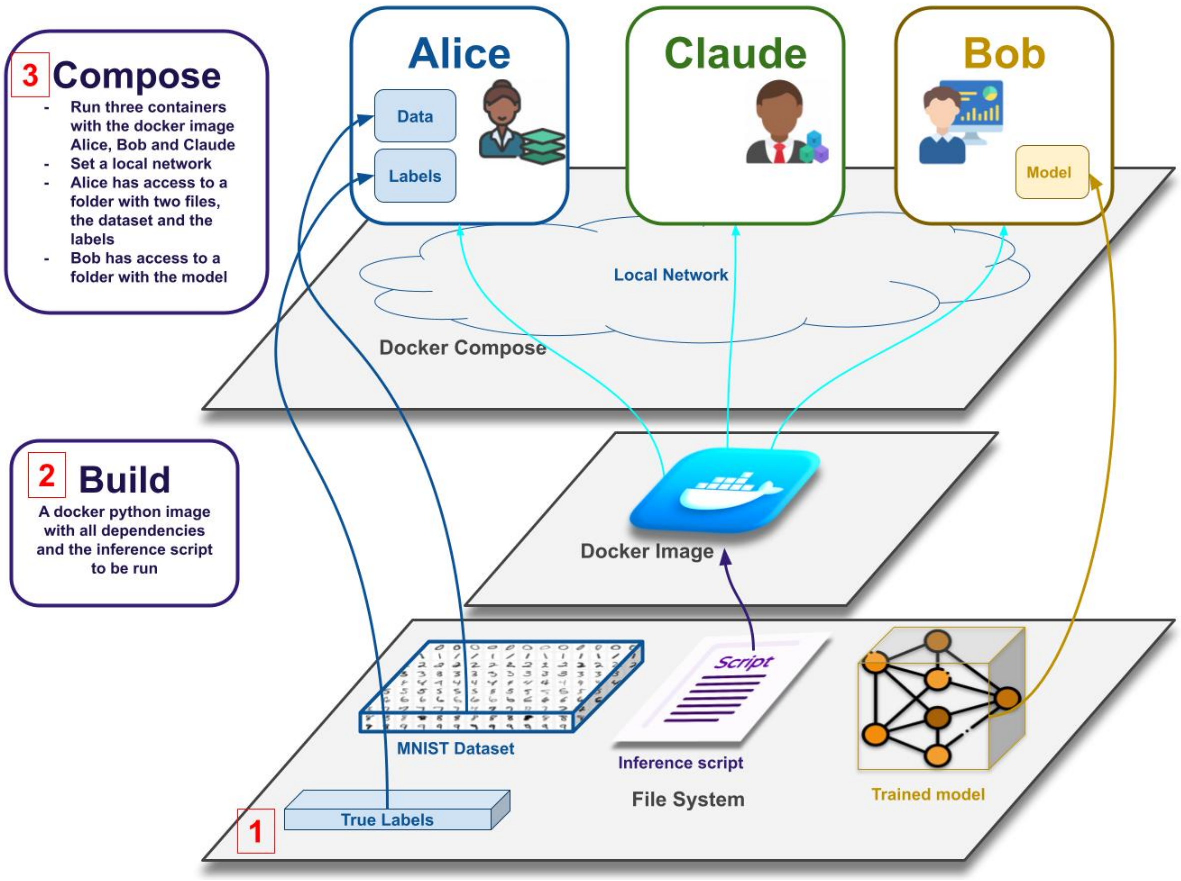
3 Compose
- Run three containers with the docker image Alice, Bob and Claude
- Set a local network
- Alice has access to a folder with two files, the dataset and the labels
- Bob has access to a folder with the model
Alice
Data
Labels
Claude
Bob
Model
Local Network
Docker Compose
2 Build
A docker python image with all dependencies and the inference script to be run
Docker Image
Script
MNIST Dataset
Inference script
File System
Trained model
True Labels
1

Figure 6.2.5: Docker setup

The benchmark of this version is shown in Table 6.2.2, which shows the average time (in seconds) of the models inference on 40 batch of 256 photo each (with average accuracy 0.9872)

Time	Alice	Bob	Claude
25% CPU	330.19	330.16	330.18
100% CPU	45.73	45.72	45.72

Table 6.2.2: the time benchmark of the model B using 3 separated machines using a quarter of a CPU core or a whole CPU core

7. Conclusion

This deliverable is part of EXTRACT project's WP2. As shown, it embodies the result of significant work done for defining data infrastructure and data mining layers that meet the ambitious requirements of EXTRACT, and further includes initial evaluations and a demonstrator. Having said that, significant work is still ahead for completing the implementation according to the laid-out design, and evaluating full end-to-end scenarios to prove the goal KPIs.

8. Acronyms and Abbreviations

- AI – Artificial Intelligence
- API – Application Programming Interface
- AWS – Amazon Web Services
- CA – Consortium Agreement
- CD – Continuous Development
- CI – Continuous Integration
- COE – Container Orchestration Engine
- CPU – Central Processing Unit
- D – Deliverable
- DevOps – Development and Operations
- DevSecOps – Development, Security and Operations
- DoA – Description of Action (Annex 1 of the Grant Agreement)
- GA – General Assembly / Grant Agreement
- GPU – Graphics Processing Unit
- HPC – High Performance Computing
- IDE – Integrated Development Environment
- IPR – Intellectual Property Right
- ISO – International Organization for Standardization
- KPI – Key Performance Indicator
- K3s – Lightweight Kubernetes
- K8s - Kubernetes
- LAN - Local Area Network
- M – Month
- MS – Milestones
- NAT – Network Address Translation
- OS – Operation System
- PaaS – Platform as a Service
- PM – Person month / Project manager
- QUIC – Quick UDP Internet Connections
- SaaS – Software as a Service
- SHA – Secure Hash Algorithm
- T – Task
- TLS – Transport Layer Security
- UDP – User Datagram Protocol
- VCS – Version Control System
- VM – Virtual Machine
- VPN – Virtual Private Network
- WP – Work Package

9. References

[1] https://openmp.org
[2] https://developer.nvidia.com/cuda-zone
[3] http://compss.bsc.es
[4] S. Chang, et.al., *"Feasibility of Running Singularity Containers with Hybrid MPI on NASA High-End Computing Resources"* CANOPIE-HPC, 2021
[5] https://aws.amazon.com/s3
[6] https://www.ibm.com/cloud/object-storage
[7] https://www.redhat.com/es/technologies/storage/ceph
[8] https://kubernetes.io/
[9] https://skypilot.readthedocs.io/en/latest/
[10] https://kubeedge.io/en/
[11] https://nuvla.io