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

D2.3 Final Release of the EXTRACT Data Infrastructure and Data Mining Framework

Version 1.0

Contract Number	**101093110**
Project Website	www.extract-project.eu
Contractual Deadline	M30, 30th June 2025
Dissemination Level	Public
Nature	Report
Author	Universitat Rovira i Virgili (URV)
Contributors	IBM, BSC, BIN, MATH, SIX, LRI
Reviewer	Observatoire de Paris (OBS)
Keywords	Data, Mining, Infrastructure

Change Log

Version	Description Change
V0.1	Initial draft of the table of contents
V0.2	Adding partners contributions
V0.3	First Review
V0.9	BSC's Review
V1.0	Version to be submitted

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

Table of Contents

1. Introduction

1.1. Purpose and objectives

The deliverable at hand will delve into the details of the **Final Release of Data Infrastructure and Data Mining for EXTRACT**, which takes place at month 30 (M30) from the project start. It will then outline how EXTRACT enables achieving objective *O1: to enable the development of complex yet secure data mining workflows, composed of advanced artificial intelligence methods and big data analytics capable of effectively handling extreme data*.

Previously, this objective was partially addressed in the past deliverables D2.1 and D2.2, where 1) requirements were gathered and 2) a MVP for EXTRACT's Data Infrastructure and Data Mining was defined. The purpose of this document is to establish the state of the platform in its final delivery (M30), prior to its evaluation, which will take place during the last 6 months of the project.

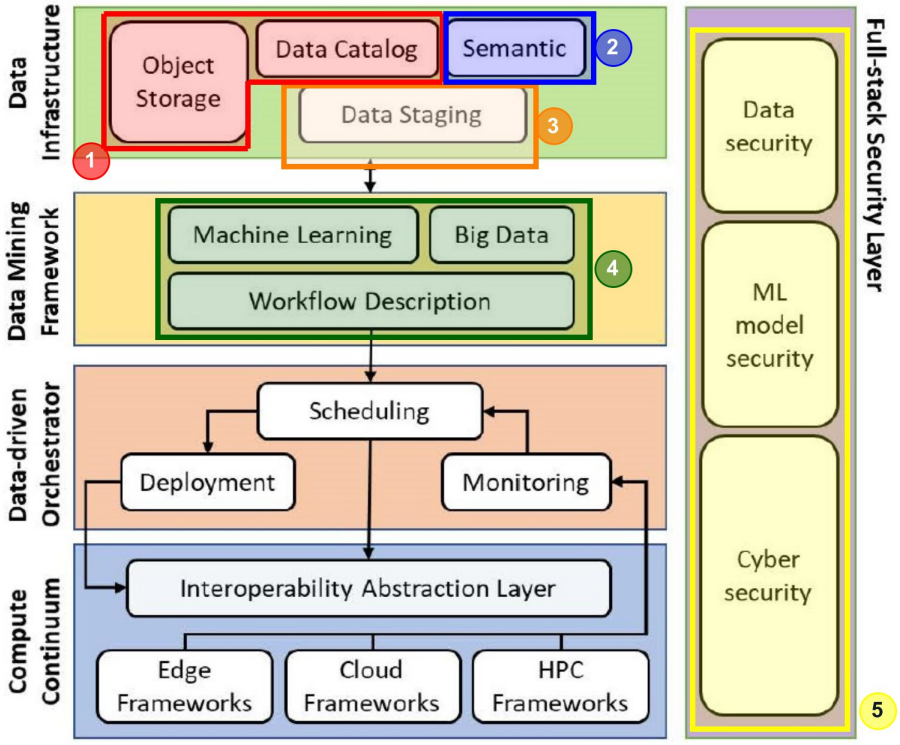

Figure 1.1 Data Infrastructure and Data Mining Layers in EXTRACT Architecture

As a reminder, Figure 1 shows the component diagram on which this deliverable focuses. For each of the different super-components that compose it, this deliverable covers 3 project tasks: *T2.2 - Data Infrastructure and Management*, *T2.3 - Data Mining Framework*, and *T2.4 - Data Security*.

The **objectives pursued** with the Final Release deliverable of EXTRACT's Data Infrastructure and Data Mining are several, as listed below:

- The **collection and presentation of the results** in a document that gathers contributions from WP2, also serving as justification of compliance with the objectives defined within the current work package.
- The **establishment of a final version** of the platform, reflecting the state of its various components prior to the project's last evaluation cycle.
- **Preparation of the tools for their upcoming evaluation** (the last 6 months of the project). By having the tools ready, the stage is set for their evaluation, so their capabilities for processing extreme data can be validated.

1.2. Relationship with other WPs and Deliverables

Deliverable	Task	Relation
D1.2	T2.1	Data infrastructure and data mining framework requirements
D2.2	T2.3, T2.3, T2.4	First release of the EXTRACT data infrastructure and data mining framework
D3.3	T3.2, T3.3	Final release of the data-driven orchestration and monitoring
D4.3	T4.2, T4.3	Final release of the Compute Continuum

Table 1.1. Relationship with other WPs

1.3. Document structure

The document is structured as follows:

- Section 1 (this one) introduces and motivates the document.
- Section 2 gives a big picture of the different components evolving the platform.
- Section 3 delves into the details of the Data Infrastructure layer (T2.2).
- Section 4 focuses on detailing the Data Mining Framework (T2.3).
- Section 5 addresses how the security layer is integrated into the framework.
- Section 6 concludes the document.

2. Big picture: Final Release of Data Infrastructure and Data Mining Framework

Focusing exclusively on the components that are part of the Data Infrastructure (grouped in the green box in Figure 1), we show in greater detail the role each framework component plays in Figure 2.

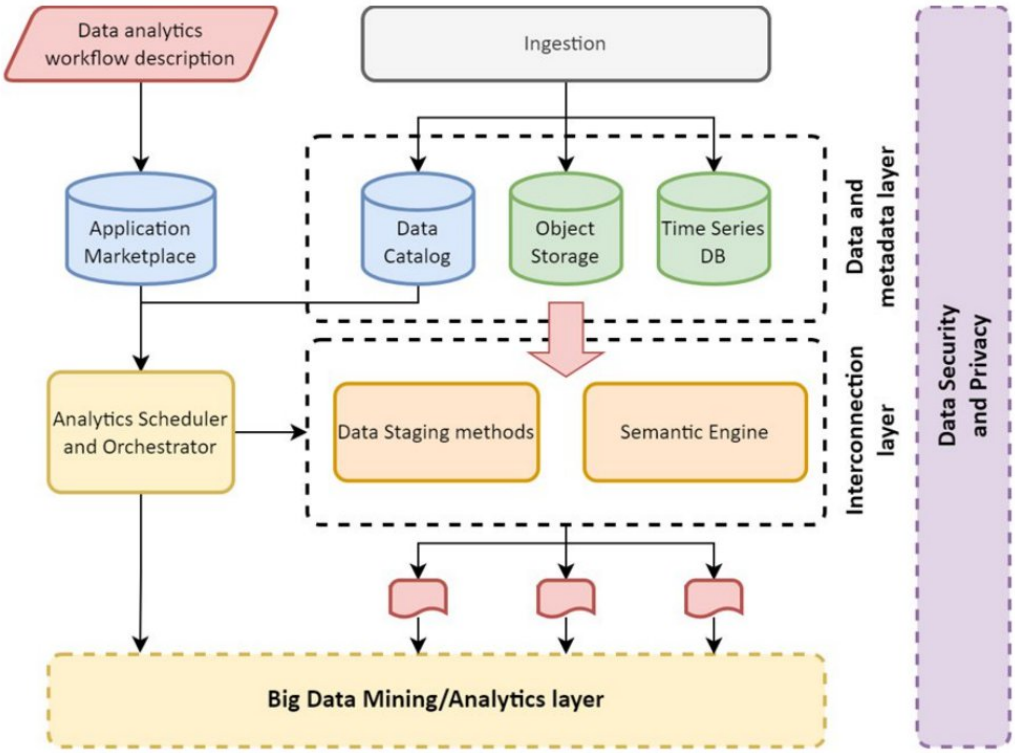

Figure 2.1 EXTRACT Data Management: Data Infrastructure (in detailed view), and interconnections with Data Mining and Data Security & Privacy

Figure 2 illustrates how, from a high-level perspective, the components forming the framework have not changed compared to the First Release of the Data Infrastructure. Although the diagram of these components was already presented in the First Release deliverable (D2.2), it is necessary to reintroduce them here because immediately after their presentation, the implementation of each component will be detailed.

- **Data ingestion**: Responsible for incorporating data into the EXTRACT platform. It covers both the transfer of data to the system and associated preprocessing tasks, including metadata generation and extraction.
- **Data and metadata layer**: Includes the solutions responsible for storage within the platform. It is based on a data lake combining massive storage in Object Storage and temporary data in a Time Series Database (Time Series DB). The Data Catalog centralizes metadata and enables its discovery, differentiating between identification metadata (name, origin, traceability) and operational metadata (indexes, metrics, procedures, etc.).

- **Semantic layer**: Facilitates intelligent data integration and retrieval by defining logical relationships between datasets, closely linked to the application domain.
- **Data staging layer**: Provides a scalable and elastic solution to prepare data from the data lake before processing. It includes tasks such as partitioning, filtering, and Extract/Transform/Load (ETL), dynamically adjusting computing resources based on data volume and the needs of mining engines.
- **Data mining layer – framework integration and workflow definition**: Constitutes the core of data processing in EXTRACT. It manages the definition of workflows and interaction with various analytics and machine learning frameworks.
- **Security, privacy, and data integrity**: Security is applied transversely across the entire platform. This layer includes mechanisms and tools to guarantee the protection, privacy, and integrity of data throughout all stages of its lifecycle.

3. Data infrastructure

As indicated in the previous section, Data Infrastructure maintains the same overall architecture like in the D2.2 previous deliverable. Although the component map and their relationships were well defined previously in deliverable D2.2, each component may present changes compared to the First Release (most likely incremental, and some may amend or modify what was specified in the First Release). In any case, the Final Release (this document) defines the state of EXTRACT's Data Infrastructure platform ahead of its evaluation

3.1. Data and metadata layer

Object Storage has been successfully integrated with the global data catalogue. This integration ensures that data ingested through the object storage layer can be immediately tracked, described, and made discoverable via the metadata catalogue. This step marks a key milestone in establishing an end-to-end data flow across EXTRACT components.

Following the initial architecture described in D2.2, the Data & Metadata Layer has remained a central element of the EXTRACT platform, providing the mechanisms for storing, describing, and organizing data in a way that supports interoperability across providers and applications.

The three core components — *data-object*, *data-record*, and *data-set* — have been further consolidated to ensure coherence in how data is described and accessed:

- **data-object** continues to represent individual data resources stored in S3-compatible buckets. It provides the abstraction layer required to decouple data storage from the consuming services or applications.

- **data-record** extends this by enabling rich metadata annotation of *data-object*s. This remains key to enabling domain-specific discoverability, facilitating cross-provider data reuse.

- **data-set** supports flexible grouping of resources, which has been instrumental in exposing relevant data collections for various use cases.

The workflow introduced in D2.2 — where *data-objects* are created, annotated via *data-records*, and grouped into *data-sets* — has been validated as an effective mechanism for organizing and retrieving data. These structures form the foundation for a unified metadata catalog and contribute to the overall goal of supporting FAIR data principles.

Integration with the **PER use case** has been successfully completed. Data generated within this context is now properly tracked and exposed via Nuvla, allowing seamless association of data types with processing applications. This confirms the viability of the metadata framework in supporting real-world data flows and application orchestration.

Work in Progress

While the PER integration validates the current metadata model and tooling, ongoing work is being carried out to support the **TASKA use case**. Compared to PER, TASKA introduces additional complexity in data handling, metadata mapping and data life cycle monitoring (also known as "provenance") , requiring adaptations at both the integration and data ingestion layers. This work is still in progress, with efforts focused on aligning TASKA-specific data flows with the established Data & Metadata Layer structures. The evaluation period that will that place in the last 6 months of the EXTRACT will be focused on defining the specifications required by TASKA for the data catalog as it is indissociable from the scientific evaluation of the products.

3.2. Semantic layer for the Urban Digital Twin

A core component of the Urban Digital Twin (UDT) is its **semantic layer**, which enables structured, interoperable, and machine-readable representation of urban knowledge. The semantic layer integrates multiple urban data sources—including road networks, mobile device locations, and events—into a unified ontology-based model, allowing advanced reasoning, querying, and integration with external datasets (e.g., Copernicus EMS).

Ontology Design
The semantic layer is built upon a modular ontology that integrates:

- **OTN (Open Transport Network)**: for road and node structures
- **SOSA/SSN**: for sensor and observation modeling (devices, traffic)
- **GeoSPARQL**: for geospatial representation (points, lines, polygons) **Schema.org and TIME ontology**: for event modeling (e.g., fire, flood, with spatial and temporal context.

The ontology is structured into **four macro-classes**:

1. **Road** – modeled via OTN:Road_Element and nodes
2. **Observations** – real-time and historical device and traffic observations using SOSA
3. **Features** – spatial entities such as buildings, bridges, greenfields
4. **Events** – dynamic events (e.g., fire, flood), with start/end times and geolocations

Graph Storage Strategy
To improve SPARQL query efficiency and scalability, the semantic layer adopts a **multi-graph architecture** in Virtuoso:

- realtime_graph: Continuously refreshed every 10 seconds with the latest device, traffic, and event data.
- historical_graph: Persisted observations every 15 minutes for long-term analysis and model training.

Reasoning in Virtuoso
Virtuoso's built-in **RDFS and partial OWL reasoning** is enabled to infer new facts from the existing RDF triples. This includes:

- Inferring class hierarchies (e.g., DeviceObservation as a subclass of SOSA:Observation)
- Resolving relationships via inverse properties (e.g., between Observation and Sensor)
- Enabling spatial inference by combining GeoSPARQL functions and topological relations
- Inferring temporal logic using time:hasBeginning, time:hasEnd, and overlapping intervals

This reasoning supports enriched queries such as:

- "Which devices are in roads affected by ongoing events?"
- "Which nodes are currently under high traffic pressure AND near a danger zone?"

Data Flow and Integration
Data is continuously consumed from Kafka topics (device_state, road_state, node_state) and mapped into RDF triples using a dedicated service (**DataSource2ttl**). These are written into Virtuoso using SPARQL Update queries, maintaining both real-time and historical views.

Benefits

- Structured and extensible knowledge base
- Improved query performance through graph separation

- Reasoning-powered queries for higher-level insights and alerts
- Foundation for semantic interoperability and external system integration

3.3. Data staging

To control and optimize data staging across the cloud-edge continuum, we have developed a set of scalable and elastic mechanisms to efficiently prepare data from storage systems to compute platforms. As a key contribution, we extended the Lithops [5] framework with a new component called **Flexecutor**, which introduces data-driven smart provisioning capabilities. This extension enables the framework to dynamically optimize data staging strategies based on the specific requirements of each application. In the context of the TASKA use cases, we applied these enhancements to the ingestion of Measurement Sets (MS) files via the new **Dataplug** plugin, implementing a fine-grained partitioning strategy that minimizes unnecessary data movement while significantly accelerating the data loading process.

3.3.1. Flexecutor: enabling smart provisioning

Smart provisioning motivation

Achieving efficient and scalable processing of large and complex datasets requires intelligent use of computational resources across heterogeneous environments as cloud, edge, and HPC, composing the continuum. This is particularly important in data-intensive workflows that must be not only performant but also cost-aware.

Distributed data processing frameworks typically rely on static partitioning strategies and fixed resource configurations. However, the optimal execution plan often depends on a combination of factors: the data volume and format, the specific computational task, and the behavior of the underlying infrastructure. In serverless environments, such as the one proposed in EXTRACT, different worker configurations (e.g., CPU/memory profiles) can lead to significant variability in throughput, making manual tuning both inefficient and impractical—especially under extreme data conditions.

To address these challenges, we introduce a smart provisioning mechanism, implemented in EXTRACT as Flexecutor, a solution built on top of Lithops. This system automates the selection of worker configuration and data chunk size for each pipeline stage by defining the different strategies pursued by the user.

Flexecutor overview

Flexecutor is a modular and extensible smart provisioning framework built on top of Lithops. While Lithops provides the core abstraction for running stateless functions in parallel across the cloud (the map() operation), Flexecutor introduces the higher-level layer that simplifies the definition, modeling, and optimization of multi-stage workflows.

It abstracts not only the execution but also the configuration, performance modeling, and optimization of tasks, allowing users to define data pipelines as composable stages, which perfectly align with TASKA use case C initial specifications. Each stage is independently configurable in terms of resource allocation, concurrency, and I/O behavior, enabling both flexibility and control.

Each stage of the computation pipelines can be processed with different parallelism degrees and different assigned resources. This is the main purpose of Flexecutor, to provide built-in support for modeling, prediction, and tuning.

Data flow abstractions

To manage I/O and data dependencies between stages, Flexecutor introduces two key abstractions: FlexInput and FlexOutput, both based on a common FlexData class. FlexInput controls how data is received by a stage—through scatter, broadcast, or chunked strategies—while FlexOutput defines how results are structured and passed downstream.

FlexData handles shared concerns like storage access points, local caching, and data partitioning, ensuring consistency and efficiency across stages. By cleanly separating data movement from computation, these abstractions make implemented workflows more modular, scalable, and easier to maintain.

Flexecutor life-cycle

Flexecutor integrates a self-adaptive performance modeling loop to optimize the execution of each stage in a workflow, which is a key feature for scalability of real-life applications on TASKA data (scaling from 1:1000 in term of data volume). This loop enables automatic tuning of resource configurations based on empirical performance data, reducing the need for manual adjustments. It consists of four main steps:

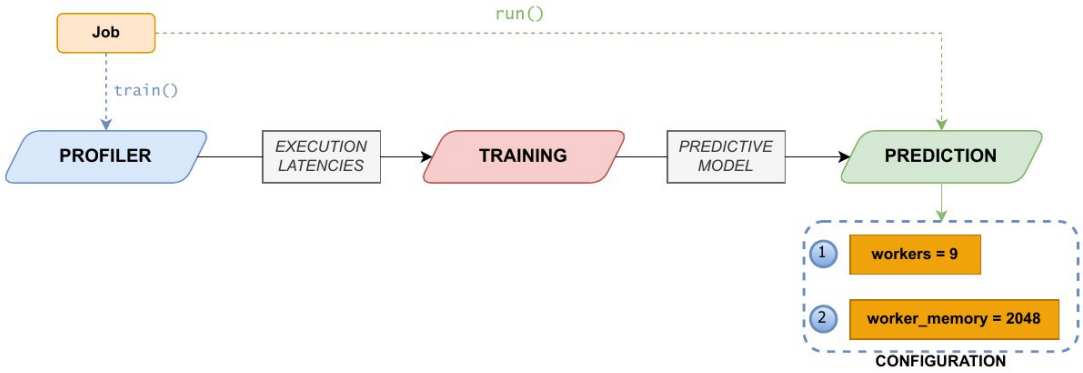

Figure 3.1. Diagram of Flexecutor operative workflow

- Profiling
  For each stage, Flexecutor runs multiple executions across different resource configurations (e.g., varying CPU, memory, number of workers). During this process, it collects fine-grained statistics, including latency and execution times for each task.
- Training
  Using the profiling data, Flexecutor trains a performance model per stage. These models—stored and retrieved per execution—capture the relationship between resource settings and stage performance. The training is handled via stage-specific perf_model objects.

- Prediction
  Once trained, these models can be used to predict execution times for unseen configurations. This enables fast, low-cost evaluation of potential setups without rerunning full experiments.
- Optimization
  Flexecutor applies optimization algorithms to search the configuration space and find the best resource allocation for each stage. Objectives may include minimizing latency, cost, or maximizing throughput.

In the final delivery of the platform, and concretely in the TASKA-C use case, Flexecutor (which operates on top of Lithops) integrates naturally with COMPSs (see Section 4.3), with each tool handling its own responsibilities: Lithops manages the deployment and control of parallel workers, COMPSs handles the orchestration of computation stages (i.e., the Lithops executors), and Flexecutor is in charge of applying all the necessary logic for optimizing the computational pipelines.

Extensible design, multiple implementations

Flexecutor supports the integration of multiple advanced scheduling and optimization strategies, many of which are adapted from prior research systems. These strategies are encapsulated as modular and interchangeable components. Flexecutor consolidates the logic of diverse smart provisioners into a single, unified tool, making it the first reusable solution for smart provisioning—addressing a significant gap, as such provisioners had not previously been integrated with any computing framework. Currently, Flexecutor incorporates four distinct smart provisioning strategies derived from research: Caerus [1], Ditto [2], Orion [3], and the latest Jolteon [4] heuristic. Thanks to its extensible architecture, Flexecutor can easily integrate future smart provisioning techniques, enabling continuous evolution and adaptation to emerging scheduling requirements.

Towards multi-backend serverless resource provisioning

We have adapted Flexecutor to support multi-backend execution (by adding a new heuristic), where different stages of a workflow can run on distinct compute substrates (e.g., cloud services like Function-as-a-Service, containers, or VMs; or compute environments across the compute continuum). This would allow us to exploit tradeoffs across backends — for example, using memory-rich containers for stages that exceed the limits of modest hardware, while keeping other stages in lightweight environments. By aligning backend selection with stage-level resource profiles, we believe this hybrid provisioning could yield improvements for different application targets across the compute continuum.

3.3.2. Measurements Set on-the-fly partitioning by Dataplug

To enable efficient, scalable, and adaptive processing of large MeasurementSet files (we remember that they are the TASKA use case inputs), we need to move towards dynamic, on-the-fly partitioning. Unlike static approaches, dynamic partitioning allows us to generate and access data slices as needed, directly within the execution workflow. This unlocks key advantages: reduces unnecessary data movement, and supports lighter-weight, parallel workers. Crucially, it makes adaptive scheduling and smart provisioning feasible, since data access patterns can be tailored in real-time without reprocessing the full dataset.

Static partitioning is inefficient

Until now, partitioning was handled through a static, monolithic process. The entire MeasurementSet (MS)—being a directory-based format—had to be zipped, downloaded, unzipped, and then fragmented using CASA[1] tools into smaller, sorted chunks. Each resulting chunk was then zipped again and uploaded to object storage (see static partitioning on Figure 3.2). This workflow duplicated data and introduced unnecessary large-scale data movement, resulting in substantial overhead. It also imposed strict memory constraints, as a worker or local process needed to load the full MS into memory to process it—limiting scalability and requiring heavyweight execution environments. While retrieving preprocessed chunks from object storage was relatively fast, the core bottlenecks—monolithic preprocessing and sequential I/O—remained, limiting the effectiveness of parallelism and making the approach unsuitable for distributed or cloud-native systems, such as the ones necessary for providing smart provisioning solutions.

Towards on-the-fly partitioning

To overcome the limitations of static partitioning, we adopt Dataplug, a framework designed specifically for efficient, on-the-fly processing of large scientific datasets already residing in object storage. Rather than relying on heavyweight preprocessing pipelines and costly full rewrites of the dataset, Dataplug enables dynamic, read-only access to partitioned views of the data without duplication or costly up-front transformation.

At its core, Dataplug introduces a format-aware indexing mechanism that preprocesses each dataset once to generate lightweight metadata (see dynamic partitioning on Figure 3.2). This metadata enables the definition of fine-grained, dynamically-sized partitions on demand. Crucially, these partitions are resolved at access time: each slice knows exactly what byte ranges to retrieve from object storage in order to reconstruct itself, resulting in effectively zero-cost slicing.

[1] Common Astronomy Software Applications, https://casa.nrao.edu/

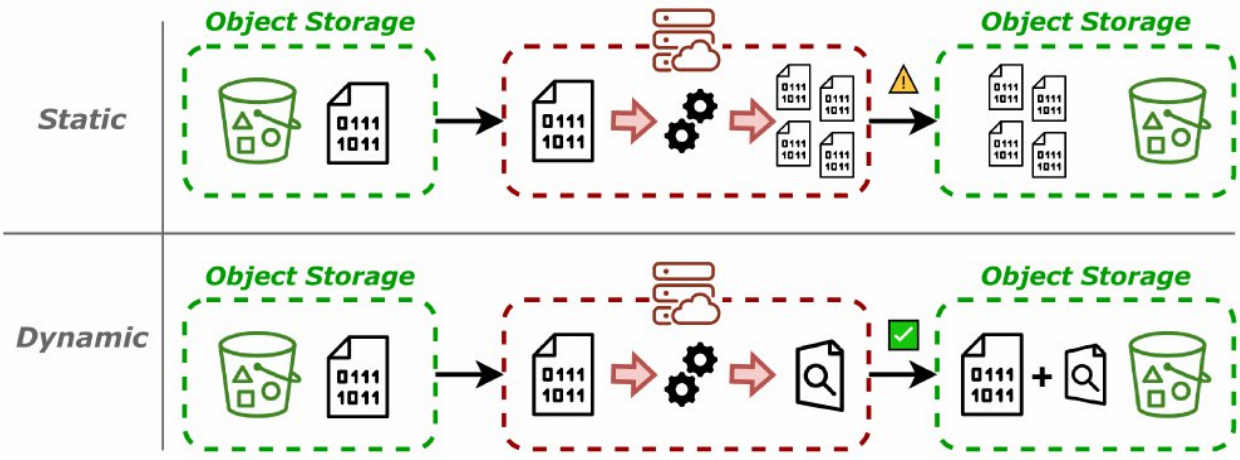

Figure 3.2. Static partitioning vs Dataplug preprocessing

This allows distributed workers to fetch partitions directly and independently, avoiding monolithic preprocessing and unnecessary data movement. Since the original data remains untouched and is not rewritten, this model is not only highly scalable (which is a one of the strong specifications of TASKA), but also cost-effective, especially for petabyte-scale datasets. Furthermore, it fully leverages the high-bandwidth nature of object storage, making it well-suited for cloud-native and elastic processing environments. Next, a diagram of MS on-the-fly partitioning with Dataplug is offered:

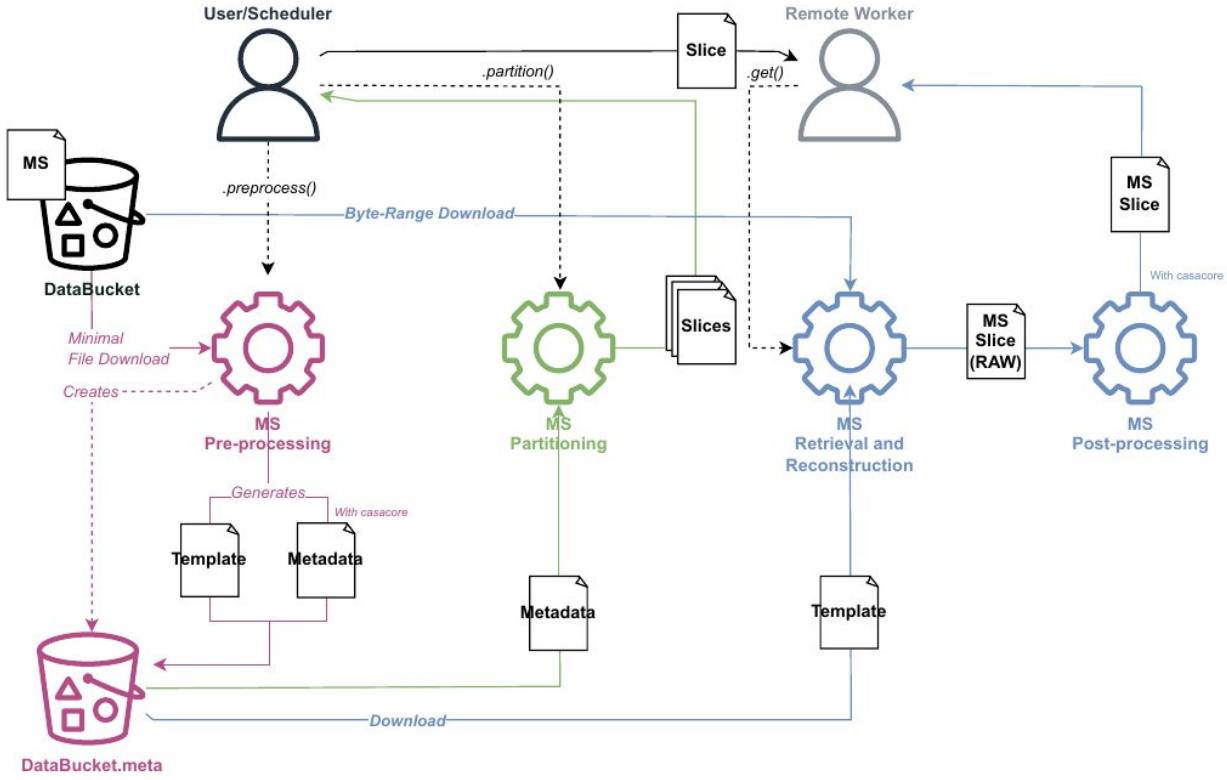

Figure 3.3. Dataplug workflow for Measurement Set ingestion

MS implementation

To make both regular execution and resource-aware provisioning more efficient, we developed a custom Dataplug plugin for the MeasurementSet (MS) format (see Figure 3.3).
The process starts with a lightweight inspection of the original MS, downloading only a minimal subset of files needed to reconstruct a metadata-rich template. This template captures the dataset's structure and content layout, enabling valid MS slices to be reconstructed directly from object storage—without requiring a full download.
This approach offers several key advantages. Preprocessing is performed only once and can even be triggered when the MS is first uploaded to object storage. From that point on, slicing becomes a zero-cost operation, as all metadata and structure are already in place. This avoids redundant data movement and eliminates the need to duplicate or rewrite large datasets.
To support this functionality, we conducted an in-depth study of the usual construction of the MeasurementSet's storage model and the Casacore[2] Table Data System (CTDS). The MS format supports a variety of storage managers, each with different I/O characteristics and format constraints. Among the most relevant for our case are:

- **StandardStMan** (Standard Storage Manager) is the default and simplest option. It stores data in equally sized buckets within the table.f<i> file. This structure is predictable and is widely used for scalar numeric data and small arrays. The small size of the data suggests that we may not need byte-level processing and retrieval. Instead, we can retain entire files and process them using Casacore operations.
- **IncrementalStMan** is optimized for data that changes infrequently and benefits from run-length compression. It is heavily used in scalar columns like FLAG and WEIGHT, where values often repeat across rows. The inherent compression of this format implies that we may not need to deeply deconstruct how this storage manager works at the byte level.
- **TiledStMan** (and its variations) organizes array data into multidimensional tiles for more uniform access performance across axes such as time, frequency, and baselines. This is conceptually similar to the chunked storage model used by the high-performance data management suite HDF5. This is where we focused most of our efforts. By design, most MeasurementSet files store large volumes of data using this storage manager, both for improved I/O and easier access.

Each **TSM** defines buckets and data blocks based on the data type (e.g., boolean, integer) and its relative size compared to the bucket. Metadata about the files stored under this manager can be retrieved using Casacore tools to check columns and further validated with overall file-level checks (e.g., total file size, number of rows, file extensions).

Each storage manager writes to its own table.f<i> file, and in some cases, auxiliary files—such as those for indirect arrays—are also used. The result is a highly fragmented but logically consistent layout, made coherent by metadata embedded in both the main data files and auxiliary files. Each column may be stored in a different physical format—usually as byte streams—optimized for its specific access pattern and update frequency.

[2] https://github.com/casacore/casacore

We also analyzed which columns are commonly present across most MeasurementSet (MS) files received from observatories and identified which of them could be addressed at the byte level. This analysis was crucial for determining how to perform slicing operations efficiently. The more we had to rely on Casacore for basic operations, the slower the overall post-processing became. For frequently accessed scalar columns (e.g., TIME, ANTENNA1, ANTENNA2... that are considered as "constants" for a given dataset containing multiple MS files), we retained their structure, since they're typically smaller and post-processing time would be a larger overhead than just downloading them and applying necessary adjustments after slicing. For more complex and larger columns—especially those containing variable-shaped arrays such as DATA—we implemented a fully custom approach for byte-level partitioning and reconstruction.
By leveraging the metadata-rich template file, we preserve the full structural layout and all necessary metadata references of the original MS, while reducing the file to its minimal viable size. This template acts as a lightweight blueprint for each slice. Reconstruction of slices in individual workers is carried out using a combination of newly developed logic and selected Casacore functions, particularly those related to column manipulation and row alignment.
The resulting slices are fully compatible with CASA-based pipelines. We validated this by comparing outputs with those from traditional static partitioning—previously used in production—confirming that dynamically sliced MS files retain the necessary integrity and behavior.
Finally, this new design naturally supports parallelism: multiple workers can independently retrieve and reconstruct different slices for post-processing. This leads to significant performance gains when scaling, making the MS plugin a highly efficient solution for distributed and cloud-native environments.

4. Data Mining Framework

The architecture for the Data Mining Framework (DMF) was laid out in D2.2, and has been successfully used in both use-cases. DMF progress in this recent milestone period reflects no design changes, only implementation progress. To avoid repetitive description but keep the context, we provide a short recap of DMF architecture, followed by a description of related work in the recent period.

4.1. DMF Overview - Recap

The purpose of the Data Mining Framework is to facilitate execution of EXTRACT workflows on top of the data whose purpose is data analytics - classic ETL (Extract-Transform-Load), ML (Machine Learning - train/serve) or combination thereof. The core principle is data-centric operation, where each workflow step has some input datasets, and may produce new datasets. Each dataset being generated or introduced needs to undergo ingestion, which involves registration in the data catalog, and further operations listed previously, such as indexing, partitioning etc. Figure 1 below, taken from D2.2, shows the architecture of the DMF and how workflows execute through it. Actual workflow orchestration is outside the scope of the DMF and is discussed in detail in deliverables D3.2 and D3.3.

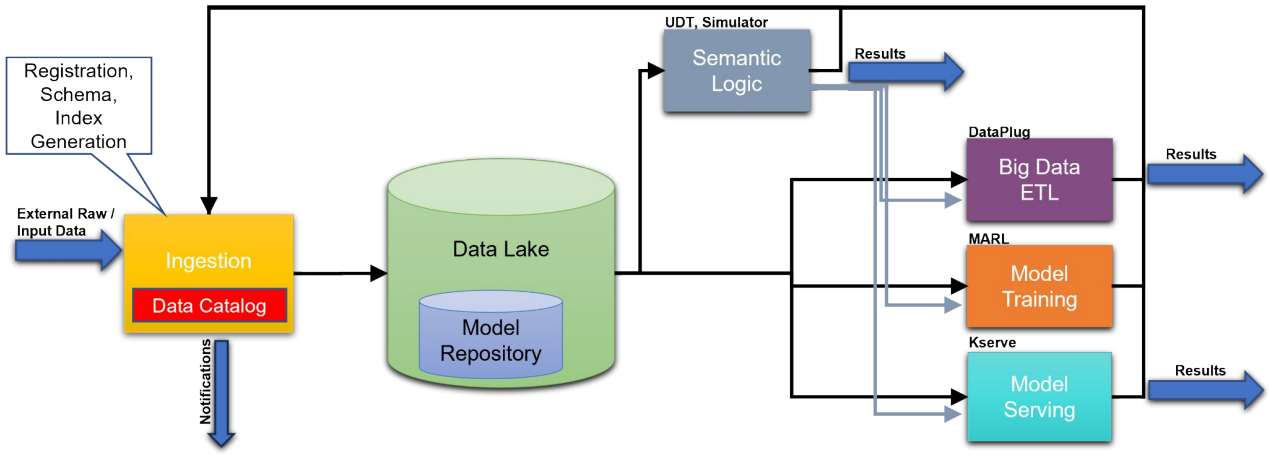

Figure 4.1. Data Mining Framework - Workflow Step Execution

As shown in Figure 4.1, the components of the DMF can be largely grouped into 3 groups, in the order of execution of a workflow step. First is the ingestion component, which is the entry for any new dataset, registering it in the data lake (and notifying optional subscribers) and doing some initial processing if needed. Next is the data lake of EXTRACT, which hosts all datasets, including ML models, which are treated as data. Last are the processing components themselves, both ETL and ML, which retrieve data from the data lake for their operation. Note that the Semantic Logic is actually an ETL component with a special role that it can also serve as an interim processing step for generating a "view" of an existing dataset as input for other processing components, and not register it as a new dataset.

4.2. Model Serving Implementation

Model serving has undergone several rounds of [re-]implementation, especially in the context of the PER use-case. Although the model serving foundation remained KServe, matching the specification described in D2.2, the actual application evolved with the use-case. After first trying to adhere to a standard model server serving a standard model file, it was realized that a custom Predictor and later a custom ServingRuntime would be a far better fit for the required model serving. Both a custom Predictor and a custom ServingRuntime allow for a specialized inference code using a platform such as PyTorch (and using a GPU if needed). However, a custom Predictor is typically meant to be used with a fixed model, so it was later replaced with a custom ServiceRuntime, that provides standardized hooks for loading a new model.

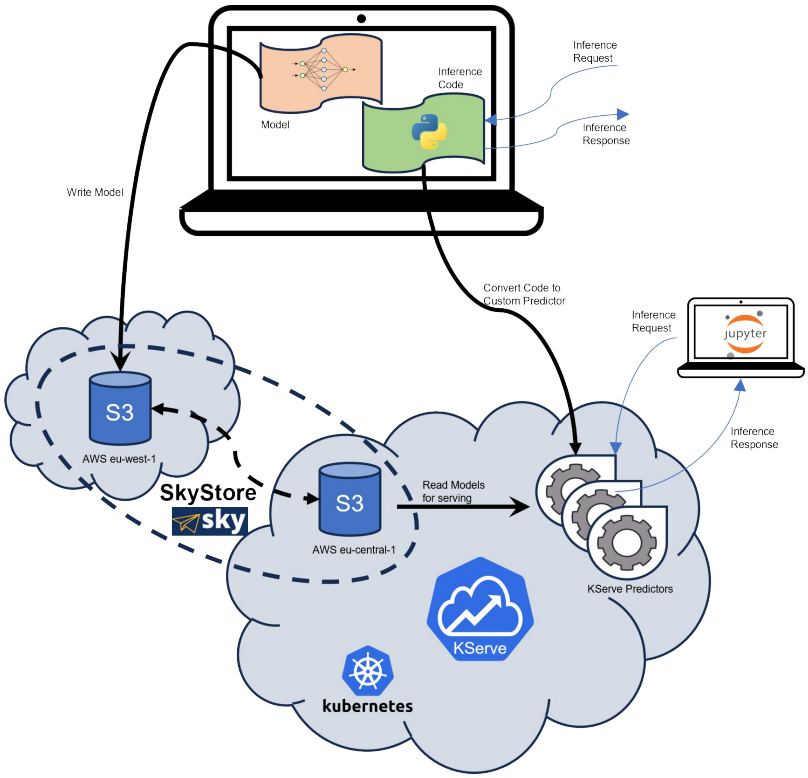

Figure 4.2. Converting custom inference to custom predictor and client across premises

As the RL model, which was trained in the HPC cluster, evolved together with a specific inference code, we were able to port the modified code into both a custom ServingRuntime and an updated client code, which were communicating as-a-service over the KServe protocol. One of the intermediate versions of this process was provided as a demonstrator in the mid-project review. Figure 4.2, taken from that demonstrator, exemplifies the process. The model is both trained and developed in the HPC premise. Modified inference code is converted into custom Predictor (now, custom ServingRuntime) with little manual labor. The model itself is propagated over SkyStore from the HPC premise to the edge premise, loaded by the updated predictor and served to clients.

4.3. COMPSs Integration with Lithops

COMPSs integrates with Lithops in a modular and transparent fashion, treating it as an external application within its programming model. From COMPSs' perspective, Lithops is simply another user-defined task that is scheduled and executed according to the global orchestration logic. However, when a Lithops task is triggered, it internally spawns a second layer of parallelism through its data-parallel execution engine, leveraging serverless backends to process many small independent subtasks concurrently.

This hybrid integration enables multi-level parallelism: COMPSs operates at the workflow level across distributed nodes, while Lithops enhances performance at the data-parallel level inside each task. This design requires no special treatment or adaptations on the COMPSs side, as the abstraction layer between the two tools ensures that the details of the internal Lithops execution remain encapsulated. Consequently, developers can write Lithops functions and invoke them from within a COMPSs application without disrupting the broader orchestration model. This layered approach enhances scalability and elasticity in data-intensive workflows, particularly in cloud-edge scenarios, while keeping the integration maintainable and loosely coupled.

4.4. Data Catalog Integration with SkyStore

The data catalog of EXTRACT is implemented using SIX Nuvla. During the DMF integration, it turned out that Nuvla could not access the S3 service provided by IBM SkyStore, which is the data backbone for EXTRACT. The issue turned out to be a missing API call in SkyStore and was fixed by IBM, as described in D4.3. The API was implemented and merged upstream into SkyStore, with data catalog integration now working across all premises.

4.5. Integrated Demonstrator WP2/WP4

The recent finalization of the DMF is showcased in an integrated demonstrator that is similar to processing in the PER use-case. The demostrator is based on the mid-project review demo that is shown in Figure 4.2. However, it features integration of the data catalog with the S3 backbone (SkyStore) for a complete data lake behavior.

In the demo scenario, we start with a file, which contains the tensor input required for inference. This file is stored in S3 using SkyStore, using one S3-proxy, and hence is hosted in the assigned “local” S3 storage. The file is also registered in the DMF data catalog. This, in return, causes a notification to be generated to subscribers. One such subscriber gets the notification and loads the file from SkyStore, but from a different S3-proxy, hence causing the file to be propagated through from the other S3 storage. The file is loaded, and then used by the subscribed to perform the inference.

The demo is presented mostly from the perspective of the subscriber that runs as a Jupyter notebook. This allows showing the key stages of the subscriber operation: starting, waiting for a notification, receiving the notification and extracting the relevant data, loading the file from the second S3-proxy, performing the inference and printing the result.

A video of the demonstrator is available in B2DROP at the D2.3 folder.

5. Data security, privacy and integrity

5.1. Data security

Data Security, at a very high level, has the following main and general aspects to be considered in the context of EXTRACT project:

- Securing "Data at rest" (i.e., storage)
- Securing "Data in motion" (i.e., communication and transfer)
- Securing "Data during use" (i.e., software application logic)

"Data at rest" refers to ensuring security (with optional integrity) of the data stored on various storage media within the diverse set of technologies and software components engaged within EXTRACT Compute Continuum flows, with various specifics related to each of the use-cases such as PER and TASKA. Up-until M30, we have successfully evaluated and tested the following "data at rest" security in a vanilla mode (i.e., not applied at present to any specific EXTRACT component):

- Kubernetes pods can leverage encrypted storage both in transit and at rest. For data in transit, pod-to-pod encryption (e.g., using Istio mTLS) ensures secure communication between pods, while encryption at rest for secrets stored in etcd is achieved through features like EncryptionConfig and external KMS providers.
    - Kubernetes "data in motion" security: Kubernetes, by default, does not encrypt traffic between pods. To achieve this, we utilize tools like Istio's mutual TLS (mTLS) and other service mesh solutions. These tools encrypt communication between pods, preventing eavesdropping and tampering. This part is addressed more in D4.3 as part of cybersecurity tasks in WP4.
    - Kubernetes "data at rest" security: Kubernetes Secrets store sensitive data like API keys and passwords. By default, they are stored unencrypted in etcd.
- Kubernetes offers features to encrypt secrets before they are written to etcd. EncryptionConfig.
    - Kubernetes allows the enabling of encryption at rest for Secrets using a KMS provider.
    - External KMS providers: We can integrate with cloud providers' KMS or tools like HashiCorp Vault for managing encryption keys and encrypting Secrets. However such KMS providers will be optionally tested/integrated, time and resource permitting during M30-M36.
- For AWS-based S3 and Object Storage, all Amazon S3 buckets have encryption configured by default, and no specific steps are required
    - For other providers this might not be true by default and required proper documentation and warning for the end-user of EXTRACT platform
- Local object storage encryption refers to the practice of encrypting data before it is stored on local object storage systems, such as those found within a single computer or a local network. This encryption can be applied either on the client-side before data is sent to the storage, or on the server-side by the storage system itself. Generally, AES-256 is a safe and good choice that is also relevant to EXTRACT deployments, nevertheless other encryption options will be evaluated from performance and usability perspectives during M30-M36.

As work-in-progress and next step, the immediate goal is to apply these tests and configurations to the integrated EXTRACT deployments (M30-M36 during Testing,

Integration, Evaluation), while measuring, fine tuning and reporting their fitness for the task and use-scae at hand. This will be performed both as part of WP2, and will be tightly coordinated with cybersecurity tasks (static and dynamic analysis, penetration testing) within the WP4.

"Data in motion" mainly deals with the transfer of data between software components of a complex software and data processing architecture such as EXTRACT. Such "data in motion" transfers usually occur over communication protocols (such as HTTPS, SSH, etc.). The choice of standardized (e.g., NIST) and/or best-practice secure protocols, as well as their secure configuration and testing, and hardening of legacy and less-secure protocols, is mainly dealt by the Cybersecurity aspects of the EXTRACT project, which are detailed in D4.3.

"Data during use" mainly deals with ensuring data integrity of essential pieces of data within massive data mass that is processed by the compute continuum flows. Such data integrity is usually implemented at the business-logic/application level. White some basic primitives have been implemented in Python to wrap basic file open()/read() into generic "integrity ensuring" open()/read() primitives, their integration into various software components of EXTRACT project is work-in-progress, and their feasibility will be tested and evaluated during the Integration, Testing, and Evaluation tasks of EXTRACT during M30-M36. Moreover, code reviews, static and dynamic analysis is continuously performed to ensure that no insecure "data during use" primitives or logic are employed (e.g., caching sensitive data into insecure /tmp files with wide/permissive/insecure filesystem access rights and without employing secure-wipe-on-close of the file after use).

5.2. 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
  - Communication only under https protocol
  - Anonymization/pseudo-anonymization of data
  - State-of-art privacy implementation (GDPR Compliant)
- Privacy by Default
  - Minimize the amount of data harvested
  - 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).
The Evacuation Mobile App (EMA) follows strict privacy by design and GDPR principles, minimizing personal data collection from the start. It only collects anonymous geolocation data necessary for evacuation guidance, encrypted during transmission. Data is kept only as long as needed for emergency response, then deleted or anonymized immediately. Users receive clear privacy notices and must give consent when required. Secure authentication limits access to sensitive features, and privacy assessments ensure strong data protection. The app enforces a zero-day data retention policy, deleting all personal and location data once the emergency ends, fully complying with GDPR's storage limitation rule and safeguarding user privacy.

Last but not least, comprehensive Privacy Policy and Data Retention Policy are in development as work-in-progress. Both policies will be tested and validated during the Integration, Testing and Validation tasks within the M30-M36 project period. The Privacy Policy is essential (and many times mandatory) for any service(s) that process any type of personally-linkable data. The Data Retention Policy is essential (and many times mandatory) for any service(s) that must retain data for legal, compliance, troubleshooting, service improvement, or other similarly legal purposes. These policies will be tailored to the PER and TASKA use-case respectively, while a generic and default "templatable" version for vanilla EXTRACT deployments will also be created, in order to facilitate privacy-compliant uptake of EXTRACT technology by other use-cases and organizations worldwide.

5.3. Model and computation protection

The need for ML model protection was raised in the PER use case. The model owner (possibly the Venice authority in a real deployment scenario) has requested to protect their model in the inference phase, i.e., to calculate the personalized evacuation route for each citizen. It has to be highlighted that the training phase of the model is performed (locally) at VEN facilities, therefore raising no particular need for protection. However, the inference computation phase may be implemented (remotely) on edge devices, which are out of the control of the model owner. In this case techniques like Multiparty Computation, preliminarily identified in the proposal as possible candidate, is not suitable since it requires the involvement of the model owner in the computation to ensure the correctness of protocol, that is not the case since the computation is on edge devices as mentioned.

Therefore, EXTRACT has started considering other protection techniques to better suit the PER inference. Homomorphic Encryption (HE) has been identified as a suitable solution for protecting functions and algorithms before moving them to another location for computation; however, for Machine Learning Models, the available implementations are still under consideration to improve the efficiency, the supported programming languages and frameworks.

Using homomorphic encryption in machine learning model privacy preserving is an application of the mathematical definition of homomorphic functions into the field of security in general. A function is called homomorphic when applied on a point *x* moves the point to a homomorphic space *H(x)* where whatever the same operations *f* are applied in both spaces, will have new points that are also homomorphically liked, $\forall f, f(H(x)) = H(f(x))$.
In the encryption terminology, an encryption is called fully homomorphic if encrypting a message *m* and then applying whatever operation or a series of operations (e.g. multiplication, addition) and then decrypting the result will have the same result as if the operations were applied directly on the clear message *Dec(f(Enc(m))) = f(m)*.

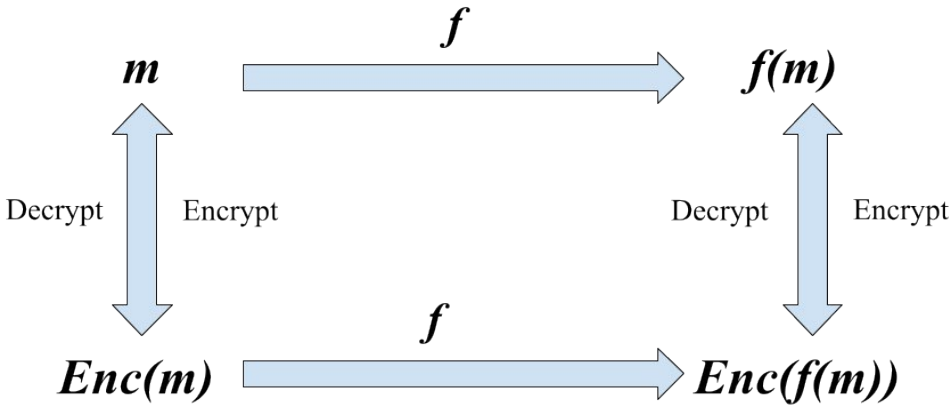

Figure 5.1. Homomorphic encryption diagram

Considering that the machine learning models, especially in deep learning based models, are a series of linear and non-linear operations applied on the input data to have the output. Homomorphic encryption is used after training the ML model, in order to preserve and protect the model weights and optionally the input data through the inference phase. Using this encryption, will guarantee the weights of the model will be confidential and can't be tempered within the use of the model in the inference computation.

The Homomorphic Encryption (HE), instead, being an off-line phase before sending the model to the edge is more suitable to this use case. The encrypted model can be used in the inference phase where the clients/users can send requests to the edge device and get the output with no information leak about the model weights.

In this period, we implemented the HE reusing the same model/data used in the MPC scenario reported in D2.2. in section "6.2. Model Protection Using MPC" and benchmarked the time and accuracy with and without the encryption shown in Table 5.3.1. In order to encrypt the model, a quantization preprocess was applied, which minimizes the bit width of the weights and the input data in order to have a smaller model. Table 5.3.1 (from left to right) shows the bit width used in the quantization, the accuracy of the original model, quantized model, fine tuned quantized model and encrypted model, then the needed time of the original model, the fine tuned quantized model and encrypted model.

Bit width	**Accuracy (%)**				**Time of one batch of 256 images inference (second)**		
	Original	Quant w/o fine tuning	Quant fine tuned	With Encryption	Original	Quant fine tuned	With Encryption
3	99.22	52.73	97.27	98.44	0.076	0.210	17
4	99.22	53.52	99.22	99.22	0.080	0.220	290
5	99.22	60.55	98.05	98.05	0.124	0.214	1264

Table 5.1. Time and accuracy benchmark of convolutional-based model on MNIST standard dataset with and without Homomorphic Encryption.

As the used model in the benchmark has two convolutional layers, it is normal to see a big difference in the needed time of the encrypted inference as the HE increases the time exponentially when using more multiplication operations within the model, which convolutional layers are known to have. In addition to using images as the dataset (28×28 pixels). However, the RL does not have any convolutional layers, so a new benchmark is needed using the encrypted RL model with the people's positions in order to better understand the computational load of the HE.

We benchmarked the HE of the RL model, in order to then integrate the solution in the PER use case inference scenario, shown in Table 5.3.2 The encrypted model has limits regarding the size of the input, this was clear when used with a map of 9276 nodes as its total size with the edges exceeds the Int16 limits (65,536).

Bit	**Map size**	**Time of one batch of 64 images inference (s)**			
		GPU			*CPU*
		Original	*Quant fine tuned*	*With Encryption*	*Original*
3	266	2.33	4.63	6.95	0.37
3	5515	41.95	44.05	23.60	15.41
3	9276	98.88	98.84	-	-

Table 5.3.2. Time and accuracy benchmark of RL model with and without Homomorphic Encryption.

Furthermore, the encrypted model was tested using the model serving locally and using the edge device. First, when used locally, the model serving was tested using the encrypted model while when using the edge device (Arm64) the library used for encrypting the model (concrete-ml) was not by default compatible. However, a compilation from source is provided by the library community and we built a compatible version of the library to be tested on the edge device.

6. Conclusion

This deliverable is part of EXTRACT project's WP2. It represents the last achievement of the work carried out to design and implement the data infrastructure and data mining layers, fully aligned with the ambitious requirements of EXTRACT. It includes the complete implementation and demonstrations showcasing the system's capabilities. While this marks a major milestone in the project, the full evaluation of the system against the targeted KPIs is planned as a next step, through comprehensive end-to-end scenario testing.

7. Acronyms and abbreviations

- API: Application Programming Interface
- CPU: Central Processing Unit
- CTDS: Casacore Table Data System
- DB: Database
- D2.1 / D2.2 / D2.3: Deliverable 2.1 / 2.2 / 2.3
- DMF: Data Mining Framework
- EMA: Evacuation Mobile App
- EMS: Emergency Management Service (e.g., Copernicus EMS)
- ETL: Extract-Transform-Load
- FaaS: Function as a Service
- GDPR: General Data Protection Regulation
- GPU: Graphics Processing Unit
- HE: Homomorphic Encryption
- HPC: High Performance Computing
- I/O: Input / Output
- KMS: Key Management Service
- M30: Month 30 (mes 30 del proyecto)
- ML: Machine Learning
- mTLS: Mutual Transport Layer Security
- MS: Measurement Set
- O1: Objective 1
- OTN: Open Transport Network
- OWL: Web Ontology Language
- PER: Personalized Evaluation Route
- RDF: Resource Description Framework
- RDFS: RDF Schema
- RL: Reinforcement Learning
- S3: Amazon Simple Storage Service
- SPARQL: SPARQL Protocol and RDF Query Language
- StMan: Storage Manager
- T2.2 / T2.3 / T2.4: Task 2.2 / 2.3 / 2.4
- TASKA: Transient Astrophysics with a Square Kilometre Array
- TSM: Tiled Storage Manager
- URL: Uniform Resource Locator
- VM: Virtual Machine
- WP: Work Package

8. References

[1] Zhang, H., Tang, Y., Khandelwal, A., Chen, J., & Stoica, I. (2021, April). *Caerus: NIMBLE task scheduling for serverless analytics*. In Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21) (pp. 653–669). USENIX Association. https://www.usenix.org/conference/nsdi21/presentation/zhang-hong

[2] Jin, C., Zhang, Z., Xiang, X., Zou, S., Huang, G., Liu, X., & Jin, X. (2023). Ditto: Efficient serverless analytics with elastic parallelism. In *Proceedings of the ACM SIGCOMM 2023 Conference* (pp. 406–419). Association for Computing Machinery. https://doi.org/10.1145/3603269.3604816

[3] Mahgoub, A., Barsallo Yi, E., Shankar, K., Elnikety, S., Chaterji, S., & Bagchi, S. (2022, July). ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)* (pp. 303–320). USENIX Association. https://www.usenix.org/conference/osdi22/presentation/mahgoub

[4] Zhang, Z., Jin, C., & Jin, X. (2024, April). Jolteon: Unleashing the promise of serverless for serverless workflows. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI '24)* (pp. 167–183). USENIX Association. https://www.usenix.org/conference/nsdi24/presentation/zhang-zili-jolteon

[5] Lithops Team. (n.d.). *Lithops documentation*. Retrieved June 25, 2025, from https://lithops-cloud.github.io/