EXTRACT

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

D2.4 Data infrastructure and data mining framework validation

Version 1.0

Contract Number	**101093110**
Project Website	www.extract-project.eu
Contractual Deadline	M39, 31 March 2026
Dissemination Level	Public
Nature	Report
Author	Enrique Molina (URV)
Contributors	IBM, BSC, BINARE, MATH, SIXSQ, LRI
Reviewer	Baptiste Cecconi (OBSPARIS)
DOI	10.5281/zenodo.19208079
Keywords	Data, Mining, Infrastructure

Change Log

Version	Description Change
V0.1	Initial draft of the table of contents
V0.2	Adding partner contributions
V0.3	Internal Review - Baptiste Cecconi (OBSPARIS)
V0.4	Apply comments (review and coordination team)
V1.0	Ready to submit

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

This document, Deliverable 2.4 (D2.4), presents the final validation and evaluation of the Data Infrastructure and Data Mining Framework developed within the EXTRACT project. Following the architectural specifications and minimum viable product definitions established in previous deliverables (D2.1, D2.2, and D2.3), this report transitions the focus from framework design to empirical assessment. Delivered at Month 39 (M39), it serves as the definitive proof of concept that the EXTRACT platform successfully enables the execution of complex, secure, and highly scalable data mining workflows across the cloud-edge-HPC continuum.

The primary objective of this deliverable is to assess the performance, scalability, and efficiency of the framework's core components under extreme data conditions. To achieve this, the validation is grounded in the project's highly demanding use cases: the TASKA-C radio-astronomy pipeline and the Urban Digital Twin (UDT) mobility ecosystem.

The document systematically evaluates the storage-agnostic Nuvla Data Catalogue and the Semantic Data Ingestion Service, which together form the unified data and metadata layer. It then provides an extensive experimental analysis of the Data Staging layer—specifically the Dataplug and Flexecutor tools—demonstrating their impact on on-the-fly data partitioning and smart resource provisioning. Furthermore, it validates the Data Mining Framework through the integration of the COMPSs orchestration model with Lithops, alongside advanced model serving capabilities via SkyStore. Finally, the deliverable assesses the transversal security and privacy mechanisms that safeguard data and computational logic across decentralized premises. Ultimately, this introduction sets the stage for demonstrating how the EXTRACT framework fulfills its target Key Performance Indicators (KPIs), significantly reducing development complexity while maximizing data throughput and computational cost-efficiency.

1.2. Relationship with other WPs and Deliverables

Deliverable	Task	Relation
D2.3	T2.2, T2.3, T2.4	Specification of the Data Infrastructure and Data Mining Framework
D1.4	T1.4	EXTRACT Use cases validation
D3.4	T3.4	Data-driven orchestration validation

Table 0. Relationship with other WPs

1.3. Document structure

The document is organized as follows:

- Section 2: Architectural overview of the EXTRACT framework.
- Section 3: Validation of the Data Infrastructure, including the semantic layer and Data Staging (Flexecutor and Dataplug).
- Section 4: Validation of the Data Mining Framework (Model Serving, COMPSs, SkyStore).
- Section 5: Security, privacy, and integrity mechanisms.
- Section 6: Final conclusions.

2. Big picture of the framework

The EXTRACT framework is conceived as a comprehensive, distributed platform designed to enable extreme data mining across the cloud-edge-HPC compute continuum. While previous deliverables established the architectural baseline and minimum viable product, this final deliverable shifts the focus toward the empirical validation, performance assessment, and KPI compliance of the integrated system. The framework operates as a cohesive pipeline where raw data is seamlessly ingested, semantically enriched, intelligently staged, and processed by advanced machine learning models within a highly secure environment.

From a high-level perspective, the architecture evaluated in this document is structured into four primary pillars: the Data Infrastructure, the Data Staging layer, the Data Mining Framework, and the Transversal Security layer.

Data Infrastructure and Metadata Management. At the foundational level, the framework relies on a robust data and metadata layer that abstracts physical storage complexities from the computational applications. This layer leverages the Nuvla Data Catalogue to provide a unified, storage-agnostic representation of all data assets, ensuring seamless tracking and discoverability across heterogeneous backends like S3 and SkyStore. Complementing this is the Semantic Data Ingestion Service, which dynamically transforms streaming operational data into structured, ontology-compliant RDF knowledge graphs. Evaluated specifically within the Urban Digital Twin (UDT) use case, this semantic layer maintains both real-time snapshots and historical observation graphs using a Virtuoso triplestore, enabling advanced spatial and temporal reasoning.

Data Staging and Smart Provisioning. To bridge the gap between massive data storage and compute engines, the framework introduces a highly innovative Data Staging layer. This layer tackles the bottlenecks of extreme data processing through two synergistic tools: Dataplug and Flexecutor. Dataplug eliminates costly data duplication by performing on-the-fly, zero-copy dynamic partitioning of complex scientific formats. Concurrently, Flexecutor acts as a data-driven smart provisioner for serverless workflows. By utilizing advanced heuristics and profiling loops (such as the Jolteon provisioner), Flexecutor automatically determines the optimal resource allocation (vCPU, memory, and parallel workers) for each pipeline stage, significantly reducing both execution time and cloud infrastructure costs.

Data Mining Framework. The processing core of EXTRACT is the Data Mining Framework (DMF), which orchestrates the execution of analytics and machine learning workloads. Its validation demonstrates a successful multi-level parallelism approach through the integration of the COMPSs programming model with the Lithops serverless engine. This hybrid orchestration allows complex, distributed workflows to scale efficiently. Furthermore, the DMF incorporates an advanced Model Serving architecture and guarantees that computational tasks have agile, scalable access to the data lake through robust integrations with IBM's SkyStore.

Security, Privacy, and Integrity. Wrapping the entire architecture is a transversal security layer. Because the framework operates across decentralized premises, robust mechanisms are evaluated to ensure data privacy, operational security, and the protection of both machine learning models and computational logic.

Ultimately, the successful integration of these components is validated through EXTRACT's highly demanding scenarios—namely, the TASKA-C radio-astronomy pipeline and the UDT mobility ecosystem—proving the framework's capability to reduce development complexity while maximizing processing throughput.

3. Validating The Data Infrastructure

3.1. Data and metadata layer

The data and metadata layer in EXTRACT is implemented through the Nuvla Data Catalogue, which provides a unified abstraction for registering, describing, discovering, and accessing data assets independently of the underlying storage back-end. The role of this layer is to decouple the physical placement of data from the logical view exposed to applications and workflows, thereby enabling the same data management model to be used across cloud, edge, and hybrid environments.

The Nuvla Data Catalogue is structured around three complementary resource types. First, data-object resources represent the stored objects themselves and manage their lifecycle in the underlying object storage. Second, data-record resources provide the searchable metadata associated with those objects. Third, data-set resources define dynamic collections of catalogue entries by means of filters, making it possible to group related data without physically duplicating them. This separation of concerns is particularly important for EXTRACT, where large volumes of heterogeneous data must be stored efficiently while remaining easy to search and reuse across the different use cases.

In the EXTRACT integration, the storage layer is exposed to Nuvla as an S3-compatible infrastructure service. Access to the storage back-end is configured through infrastructure-service resources and the corresponding credentials stored in Nuvla. This allows the data catalogue to remain storage-agnostic while still supporting concrete back-ends such as SkyStore or other S3-compatible services. As a result, applications interact with a stable catalogue model, whereas changes in the storage provider or deployment environment only require reconfiguration of the infrastructure service and credentials rather than modifications to the application logic.

The metadata model used in EXTRACT combines the standard Nuvla attributes with application-specific metadata attached to data-record resources. The implementation supports namespaced metadata under the extract: prefix, making it possible to encode domain information such as observation identifiers, processing status, ingestion timestamp, file size, batch information, or other use-case-specific attributes. This approach ensures that metadata remains both extensible and semantically well scoped, while still benefiting from Nuvla's query and filtering capabilities.

This design has been validated through the data catalogue integration developed for the EXTRACT workflows. In the implemented ingestion flow, each newly ingested file is uploaded to object storage and then registered in Nuvla through the creation of a corresponding data-object and data-record. The data-record contains the searchable metadata required by downstream components, while the link to the underlying storage object is preserved through the associated data-object. Optional data-set resources can then be created to group related entries through filters, for example all raw files associated with a given observation identifier.

The validation performed in EXTRACT confirms that the Nuvla Data Catalogue can act as the common data and metadata layer across use cases. It supports storage abstraction, rich metadata registration, search and retrieval by metadata, and the construction of higher-level logical collections of data. This provides the necessary foundation for data-driven processing pipelines in which ingestion, discovery, downstream processing, and notification can all be coordinated through a consistent catalogue model.

From the perspective of the project KPIs, this layer contributes primarily to interoperability and reduced development effort. It avoids ad hoc, storage-specific handling of metadata in the applications and provides a reusable integration pattern that can be applied across EXTRACT scenarios. At the same time, it establishes the metadata basis required for later stages of the framework, including data-driven orchestration and event-based processing.

3.2. Semantic layer for the Urban Digital Twin

Overview. The Semantic Data Ingestion Service constitutes the semantic integration layer of the Urban Digital Twin (UDT) within the EXTRACT framework. Its primary objective is to transform heterogeneous operational data streams into structured, semantically enriched representations that can be stored, queried, and analysed within a knowledge graph environment.

The service consumes data from Kafka topics representing the current system state and maps these streams into RDF triples compliant with the EXTRACT ontology. The generated semantic data are persisted in a Virtuoso triplestore and organised into two complementary graph structures:

- Historical observation graphs, supporting temporal accumulation and retrospective analysis
- Real-time snapshot graphs, providing a consistent and up-to-date representation of the system state

This dual-graph architecture enables both real-time decision support and long-term analytical processing, thereby contributing to the EXTRACT objectives of data interoperability, scalability, and advanced analytics enablement.

Data Ingestion Workflow. The Semantic Data Ingestion Service operates as a continuous, asynchronous pipeline that processes streaming data from the UDT ecosystem. It consumes aggregated state information related to devices, roads, and nodes, and transforms these data into semantic observations.

The workflow consists of the following stages:

1. Kafka ingestion: The service subscribes to multiple Kafka state topics, ensuring continuous acquisition of system state updates.
2. State transformation: Incoming JSON records are parsed and semantically mapped to RDF observations in accordance with the EXTRACT ontology.
3. Graph generation: RDF triples are constructed using RDFLib and organised into named graphs, enabling structured storage and retrieval.
4. Triple store insertion: The generated RDF graphs are inserted into a Virtuoso triplestore through SPARQL update operations, ensuring persistence and queryability.
5. Notification of graph availability: Upon successful ingestion, metadata notifications may be published via Kafka, enabling downstream components to react to newly available semantic data.

This workflow ensures near real-time semantic integration of streaming data, contributing to KPIs related to timeliness and system responsiveness.

Graph Management Strategy. To address both real-time and historical analysis requirements, the ingestion service maintains two distinct graph structures.

Historical Observation Graph. The historical graph accumulates time-stamped observations associated with spatial entities of the UDT. Each observation includes:

- timestamp
- spatial location
- observed entity (device, road, node)
- measured attributes (e.g., speed, device count)

Graphs are organised using date-based naming conventions, enabling efficient temporal partitioning and scalable data management.

This structure supports:

- longitudinal analysis of mobility patterns
- reconstruction of traffic dynamics
- behavioral modelling of population flows
- generation of training datasets for machine learning

The historical graph directly contributes to KPIs related to effectiveness and analytical capability, by enabling evidence-based evaluation of system behavior over time.

Real-Time Snapshot Graph. The real-time snapshot graph provides a consistent representation of the current system state. Unlike the historical graph, it follows a **graph replacement strategy**:

- a new graph is generated at each ingestion cycle
- the previous snapshot graph is removed
- the new graph replaces the previous state

This approach ensures that only the latest observations are maintained, eliminating the need for temporal filtering and enabling efficient real-time queries.

The snapshot graph supports:

- real-time monitoring dashboards
- decision-support systems
- operational situational awareness

This component contributes directly to KPIs related to timeliness, rapidity, and operational efficiency, by ensuring that decisions are based on up-to-date system information.

RDF Mapping Service. The RDF mapping component is responsible for converting incoming JSON data into ontology-compliant RDF triples. The mapping process generates semantic entities representing:

- mobile devices
- device observations

- road traffic observations
- node traffic observations
- spatial geometries
- temporal entities

Spatial data are encoded using GeoSPARQL geometries (WKT), while temporal information is represented using the Time ontology (time:Instant). Additionally, the service resolves road identifiers by matching node pairs to the corresponding road topology, ensuring semantic consistency between streaming data and the underlying spatial model.

This semantic enrichment layer enhances data interoperability and semantic consistency, directly supporting KPIs related to integration and reuse of heterogeneous data sources.

Triple Store Integration. The ingestion service interfaces with a Virtuoso triplestore to persist and manage RDF graphs. Core functionalities include:

- execution of SPARQL queries
- insertion of RDF data into named graphs
- replacement of snapshot graphs
- chunked insertion of large RDF datasets

To improve reliability and scalability, large RDF graphs are partitioned into smaller chunks prior to insertion, mitigating SPARQL endpoint timeouts. A health-check mechanism is also implemented to ensure service availability and robustness. This design supports KPIs related to system scalability and reliability, ensuring stable operation under high data throughput conditions.

Notification Mechanism. Upon successful graph generation or update, the service can emit notifications via Kafka to inform downstream components. These notifications include:

- graph URI
- creation timestamp
- number of inserted triples
- number of processed messages
- producing service identifier

This mechanism enables event-driven processing and orchestration, allowing other components to react dynamically to data updates. It contributes to KPIs related to timeliness and system coordination, supporting near real-time processing pipelines.

Role within the Urban Digital Twin Architecture. The Semantic Data Ingestion Service serves as the core semantic integration layer of the Urban Digital Twin. It bridges streaming operational data and knowledge-based representations, enabling:

- continuous monitoring of system state
- temporal and spatial analysis of urban dynamics
- semantic querying and reasoning
- integration with visualisation and decision-support systems

By combining real-time and historical data management with ontology-based modelling, the service provides a scalable and interoperable foundation for advanced analytics.

3.3. Data staging

The data staging layer in EXTRACT is responsible for preparing data from storage systems for efficient consumption by compute platforms, and for optimising the provisioning of computational resources to process that data. As presented in D2.3, this layer is realised through two complementary tools developed in the project: **Flexecutor**, a smart provisioning framework for computational workflows, and **Dataplug**, an on-the-fly dynamic partitioning framework for scientific data. Together, they address the full data staging lifecycle: Dataplug handles the efficient preparation and partitioning of data, while Flexecutor optimises the allocation and configuration of computational resources that process it.

This section presents a comprehensive experimental evaluation of both tools, applied to the TASKA-C use case imaging calibration pipeline. We first describe the experimental setup, then diagnose the computational characteristics of the pipeline, and subsequently evaluate Flexecutor and Dataplug individually. We conclude with an assessment of the project KPIs addressed by these tools and a summary of the key findings.

3.3.1. Experimental setup

The evaluation is conducted on the **TASKA-C** use case pipeline, a radio-interferometry data calibration workflow representative of the extreme data processing challenges targeted by EXTRACT. The pipeline processes Measurement Set (MS) files — a standard data format in radio astronomy — through three sequential stages:

1. **Rebinning**: Reduces the spectral resolution of the raw visibility data by averaging frequency channels. This stage receives the largest data volume and produces a significantly reduced output.
2. **Calibration**: Applies gain corrections to the rebinned visibilities, compensating for instrumental and atmospheric effects.
3. **Imaging**: Transforms the calibrated visibilities into sky images. This is a full-reduce step that concludes the pipeline, producing an image as target output.

The experiments were executed on **AWS Lambda** as the Function-as-a-Service (FaaS) compute backend and **Amazon S3** as the object storage layer. It is important to emphasise that this setup is **fully cloud-agnostic**: both Flexecutor and Dataplug are built on top of Lithops, which provides a unified API across multiple cloud providers (AWS, Google Cloud, Azure, IBM Cloud) as well as non-cloud backends. The same pipeline can be deployed on HPC clusters, edge computing environments, or hybrid cloud-edge configurations without modifications to the application code — only the Lithops backend configuration needs to be changed. This portability is a key design principle of the EXTRACT data staging layer and ensures that the results presented here generalise beyond the specific infrastructure used for evaluation.

3.3.2. Preliminary evaluations: Characterising the TASKA-C use case

Before evaluating the smart provisioning and data partitioning tools, it is essential to understand the computational characteristics of the target pipeline. This diagnosis motivates the need for intelligent resource management and reveals the heterogeneous nature of the workload across stages.

Data Input/Output per Stage. Figure 1 presents the data volumes flowing through each stage of the pipeline for a representative ~8 GB input Measurement Set. Although the absolute volumes scale with dataset size, this example is sufficient to illustrate the data generation and reduction patterns across stages. The three stages exhibit markedly different I/O profiles.

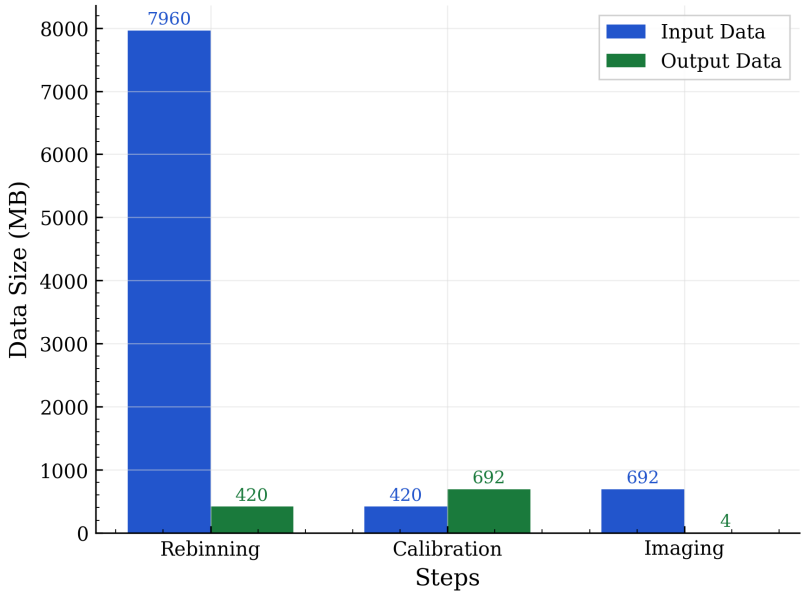

Figure 1. Data input and output sizes per pipeline stage.

The rebinning stage ingests the largest volume (7,960 MB) and performs aggressive data reduction, producing only 420 MB of output — an 18.95× reduction. Calibration slightly expands the data (from 420 to 692 MB) due to the addition of corrected visibility columns. The imaging stage achieves the most dramatic reduction, compressing 692 MB of calibrated visibilities into a 4 MB image product.

These observations have direct implications for resource provisioning: the rebinning stage is I/O-dominated and benefits significantly from parallel data access, while imaging is compute-dominated with minimal data movement. The total data movement across the pipeline amounts to approximately 10 GB of reads and 1.1 GB of writes, underscoring the importance of data transfer optimisation (see Dataplug).

Data Throughput per Stage. Figure 2 shows the computational throughput (MB/s) achieved at each pipeline stage under different vCPU allocations. The throughput measurements reveal fundamentally different scaling behaviours.

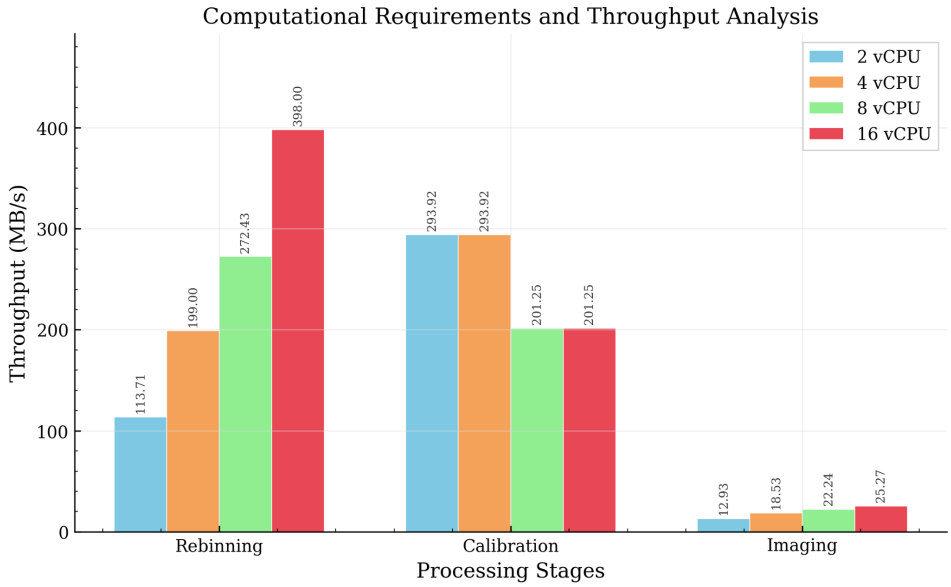

Figure 2. Computational throughput per stage under different vCPU allocations.

- **Rebinning** exhibits strong vertical scalability, with throughput increasing nearly linearly from 113.71 MB/s at 2 vCPU to 398.00 MB/s at 16 vCPU — a 3.5× improvement. This indicates that the rebinning workload is CPU-bound and efficiently scales vertically with computational resources.
- **Calibration** shows a notably different pattern: throughput plateaus at approximately 294 MB/s for lower vCPU counts and actually decreases to around 201 MB/s at higher allocations. This suggests that calibration is bounded by factors other than raw CPU power — likely memory bandwidth or I/O contention — and that over-provisioning resources (more than a single CPU per logical worker) for this stage would be wasteful.

- **Imaging** has the lowest throughput overall (12.93–25.27 MB/s), reflecting the computational intensity of this stage. While throughput does increase with vCPU allocation, the gains are modest (approximately 2× from 2 to 16 vCPU), indicating that imaging slightly scales with vertical allocation.

These heterogeneous scaling behaviours across stages are precisely the challenge that motivates smart provisioning: a single, uniform resource configuration applied to all stages would inevitably lead to either under-provisioning (hurting performance) or over-provisioning (wasting resources and increasing cost). Flexecutor addresses this by enabling per-stage resource optimisation, as evaluated in Section 3.3.

Expressivity and Development Complexity. A key design principle of the TASKA-C use case is **expressivity**: the pipeline library is architected so that domain scientists — in this case, radio astronomers — can define and execute complex multi-stage workflows using a concise, high-level client API, without needing to manage the underlying complexities of serverless orchestration, data movement, or cloud resource management.

To quantify this principle, we measured the source lines of code (SLoC, excluding blanks and comments) at two levels of the TASKA-C software stack. The **client-level code** — the script that defines and executes the complete three-stage pipeline (rebinning, calibration, and imaging), including all stage parameters, I/O bindings, and execution triggers — comprises only **167 SLoC**. In contrast, the **library internals** that this client code leverages encompass **1,347 SLoC**, distributed across several modules: workflow orchestration and configuration (475 SLoC), I/O management and serverless data transfer (276 SLoC), pipeline step definitions and execution logic (367 SLoC), and data partitioning and chunking (229 SLoC).

This yields a **complexity ratio of approximately 8:1** — for every line written by the end user, eight lines of orchestration, data management, and execution logic are encapsulated by the library. In practical terms, a domain scientist expresses a full serverless radio-interferometry pipeline without writing a single line of cloud infrastructure code, parallel execution logic, or data serialisation routines. The EXTRACT framework (Lithops + Flexecutor + Dataplug) is entirely hidden behind the library's abstractions.

This expressivity metric complements the framework-level code savings reported in Section 3.3.3 (where Flexecutor reduces boilerplate by approximately 20% compared to vanilla Lithops): while the latter measures savings *within* the orchestration layer, the complexity ratio captures the broader development effort reduction experienced by the end user of the EXTRACT-enabled application. Together, these two metrics provide a comprehensive view of KPI 1.2 (reduction in development costs) from both the framework developer's and the domain scientist's perspectives.

3.3.3. Flexecutor: enabling smart provisioning

This section evaluates Flexecutor ability to model, predict, and optimise the resource configuration of serverless workflows. The evaluation uses the TASKA-C pipeline with a ~1 GB Measurement Set dataset, profiled across a grid of resource configurations: **{1, 3, 6} vCPU** (corresponding to 1,769, 5,307, and 10,240 MB of memory on AWS Lambda, where memory scales proportionally with vCPU) and **{1, 2, 4, 8, 16} parallel workers**. This yields 15 configurations for the rebinning and calibration stages. Since imaging operates on the aggregated output of the preceding stages and cannot be parallelised in the same manner, it was profiled with a single worker under 3 vCPU configurations. Each configuration was executed at least twice to account for variance.

Per-Stage Cost vs. Execution Time Trade-offs. Figure 3 presents scatter plots of execution cost versus execution time for each pipeline stage, with the Pareto-optimal front highlighted for stages with sufficient configuration diversity.

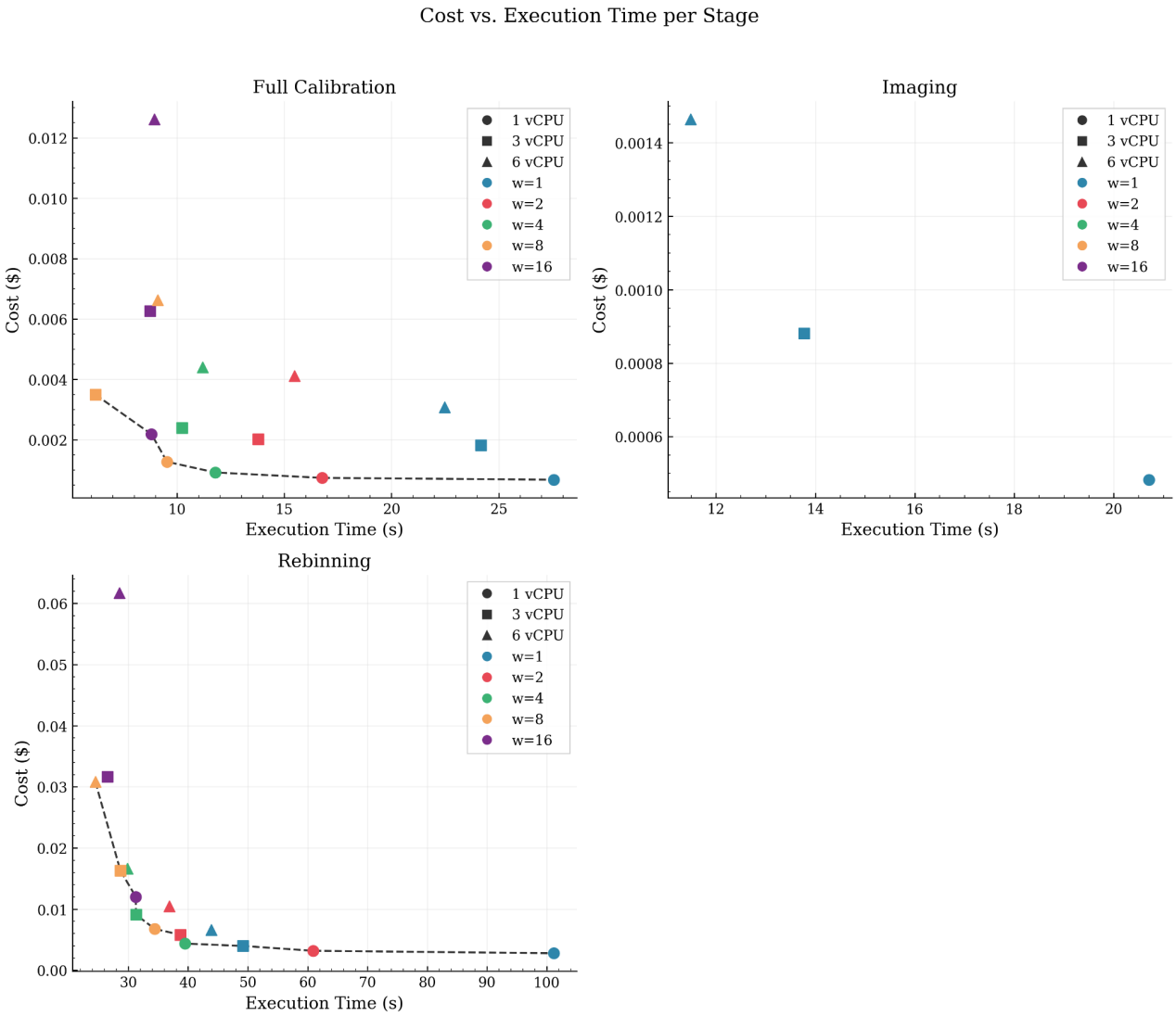

Figure 3. Cost vs. execution time per stage, with Pareto-optimal front (dashed line)

- **Rebinning** shows a clear and well-defined Pareto front spanning from the cheapest configuration (1 vCPU, 1 worker: 101.16 s at $0.00279) to the fastest (6 vCPU, 8 workers: 24.48 s at $0.03082). The 4.13× speedup comes at an 11.0× cost increase, illustrating the classic time-cost trade-off. Intermediate Pareto-optimal points include 1 vCPU / 4 workers (39.49 s, $0.00437) and 3 vCPU / 4 workers (31.27 s, $0.00910), which offer compelling compromises. The wide spread of non-Pareto configurations (particularly those with high parallelism but low vCPU, or vice versa) demonstrates that naive resource selection can be highly suboptimal.
- **Calibration** also exhibits a well-defined Pareto front, ranging from 1 vCPU / 1 worker (27.56 s, $0.00068) to 3 vCPU / 8 workers (6.19 s, $0.00350). The Pareto front is notably steeper than rebinning's, reflecting calibration's different scaling characteristics. The cheapest configuration with low vCPU achieves remarkably low cost, confirming that calibration does not require high computational resources per worker. The most cost-effective parallelisation uses 1 vCPU / 4 workers (11.78 s, $0.00092), achieving a 2.34× speedup with only 1.35× cost increase over the serial baseline.
- **Imaging**, profiled with only a single worker across 3 vCPU levels, shows a simple trade-off: execution time decreases from 20.71 s (1 vCPU, $0.00048) to 11.49 s (6 vCPU, $0.00146). The 1.80× speedup comes at a 3.04× cost increase, indicating moderate vertical scaling efficiency.

***Pipeline-Level Cost vs. Time*.** Figure 4 shows the aggregated cost and time for full pipeline executions across all tested configurations. The pipeline-level analysis reveals the compounding effects of per-stage configuration choices.

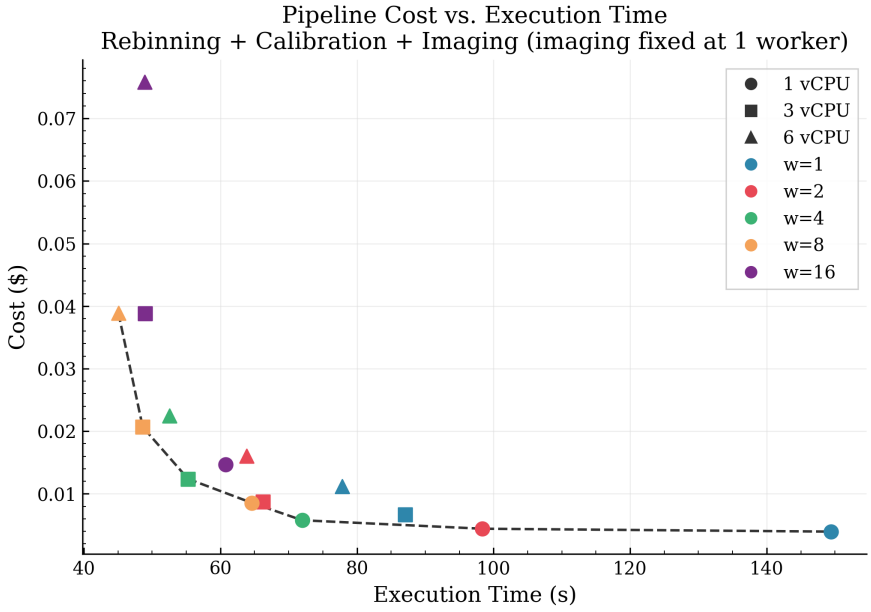

Figure 4. Pipeline cost vs. execution time (rebinning + calibration + imaging, with imaging fixed at 1 worker)

- **Best execution time**: 45.06 s (6 vCPU, 8 workers) at a cost of $0.03891
- **Worst execution time**: 149.44 s (1 vCPU, 1 worker) at a cost of $0.00395
- **Cheapest execution**: $0.00395 (1 vCPU, 1 worker) taking 149.44 s

- **Most expensive execution**: $0.07580 (6 vCPU, 16 workers) taking 48.91 s

The worst-to-best time ratio is **3.32×**, meaning the slowest configuration takes more than three times longer than the fastest. More strikingly, the cost ratio reaches **19.17×**, with the most expensive configuration costing nearly twenty times more than the cheapest. This dramatic range demonstrates the critical importance of proper configuration selection, which Flexecutor automates through its smart provisioning capabilities.

Resource Wastage Analysis. To quantify the penalty of suboptimal resource configuration, we compute resource wastage heatmaps for each stage and for the full pipeline. Wastage is defined as the percentage increase over the optimal value for a given metric (cost, execution time, or performance, where performance = 1/(cost × time)).

– **Per-stage wastage** (Figure 5) reveals significant penalties for poor configuration choices:

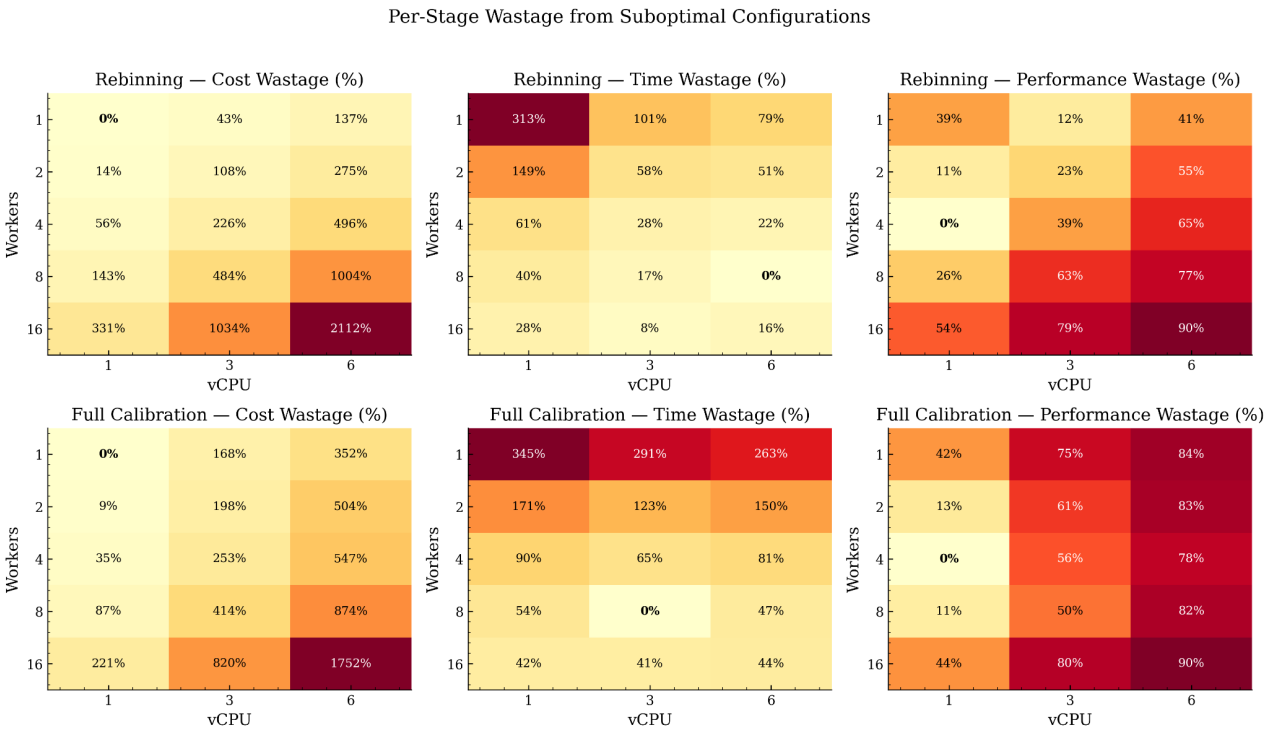

Figure 5. Per-stage resource wastage heatmaps: cost, time, and performance wastage for rebinning (top) and calibration (bottom)

- **Rebinning** shows cost wastage of up to 2,112% (6 vCPU, 16 workers versus the optimal 1 vCPU, 1 worker). Time wastage reaches up to 313% (1 vCPU, 1 worker versus the optimal 6 vCPU, 8 workers), confirming that some configurations are over four times slower than the optimum. Performance wastage reaches 90%, meaning the worst configuration achieves less than 10% of the best configuration's efficiency.
- **Calibration** exhibits cost wastage of up to 1,752% and time wastage of up to 345%. These extreme values are driven by configurations that allocate excessive resources (high vCPU counts) to a stage that does not benefit from them.

– **Pipeline-level wastage**. From Figure 5 we can aggregate the data to get the pipeline wastage view. The numbers show that the optimal configurations cluster in the mid-range of the resource grid (moderate vCPU with moderate parallelism), while the extremes — particularly high vCPU combined with high worker counts — yield the highest wastage, reaching up to 1,817% in cost and 232% in time. This underscores that the resource provisioning problem is non-trivial: neither maximal nor minimal resource allocation is optimal, and the sweet spot depends on the specific stage characteristics.

Impact of Configuration on Cumulative Execution. Figure 6 illustrates the divergence between best and worst resource configurations as the number of pipeline executions grows.

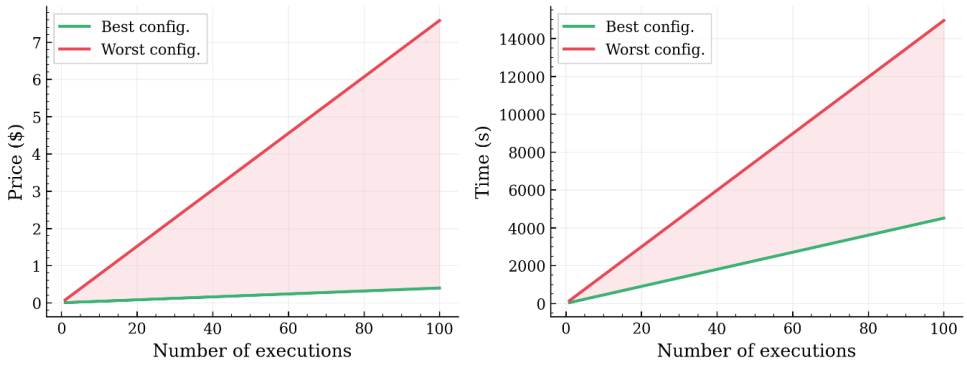

Figure 6. Impact of resource configuration on cumulative price, time, and performance over N = 1…100 executions

Over *N* = 1 to 100 executions:

- **Price**: The cumulative cost gap widens linearly. After 100 executions, the worst configuration would cost approximately $7.58, compared to $0.40 for the best — a difference of **$7.18**, following the 19× multiplier price noticed in previous heatmap. For production workloads that execute pipelines thousands of times, this translates to substantial budget impact.
- **Time**: Similarly, the total execution time diverges from approximately 4,506 s (best) to 14,944 s (worst) over 100 runs — a difference of nearly 3 hours.
- **Performance**: the gap between best and worst performance (1/(cost × time)) is 8x across the pipeline.

This analysis powerfully motivates the use of smart provisioning: even small improvements in configuration selection compound over repeated executions, yielding significant savings in both cost and time.

Smart provisioning. Flexecutor integrates four state-of-the-art smart provisioners (Caerus, Ditto, Orion, and Jolteon). For this evaluation, we focus on **Jolteon**, the most recent and state-of-the-art approach, which combines white-box and black-box modelling into a stochastic framework that uses Monte Carlo sampling to convert uncertain optimisation problems into deterministic ones.

- ***Prediction Accuracy***

After profiling and training, Jolteon generates a predictive model for each pipeline stage. Table 1 reports the prediction accuracy, measured as the mean absolute error and mean signed error between predicted and actual execution times:

Stage	*Mean abs error (%)*	*Mean error (%)*
REBINNING	5.1	0.5
CALIBRATION	28.21	10.52
IMAGING	15.39	4.29

Table 1. Jolteon prediction errors: mean absolute error and mean signed error per stage

Rebinning predictions are remarkably accurate, with only 5.10% mean absolute error and a near-zero bias (0.50% mean error). This is consistent with rebinning's regular scaling behaviour observed in the throughput analysis.

Calibration shows higher prediction error (28.21%), reflecting the stage's irregular scaling pattern — the throughput plateau and decrease at higher vCPU allocations create a more complex performance surface for the model to capture. The positive mean error (10.52%) indicates a slight tendency to overestimate execution time, which results in conservative (safe) provisioning recommendations.

Imaging achieves moderate accuracy (15.39% absolute error, 4.29% bias), adequate for a stage where only vertical scaling applies.

- ***Recommendations under Different Objectives***

Table 2 presents the resource configurations recommended by Jolteon under different optimisation bounds. The recommendations are computed for two objective types — latency bounds (maximum acceptable execution time) and cost bounds (maximum acceptable cost per execution) — at various thresholds. The tuples indicate (memory in MB, number of workers), and all recommendations are verified to satisfy the specified bound.

Latency-bound Recommendations

Bound	*REB*	*CAL*	*IMG*	*Satisfy bound?*
50s	(10614, 16)	(10614, 16)	(10614, 1)	OK
80s	(10614, 16)	(1769, 16)	(10614, 1)	OK
100s	(1769, 16)	(10614, 16)	(1769, 1)	OK
180s	(10614, 1)	(1769, 1)	(1769, 1)	OK

Cost-bound Recommendations

Bound	*REB*	*CAL*	*IMG*	*Satisfy bound?*
$0.01	(1769, 16)	(1769, 16)	(1769, 1)	OK
$0.05	(1769, 16)	(1769, 16)	(10614, 1)	OK
$0.10	(10614, 16)	(1769, 16)	(10614, 1)	OK

Tuple format: (memory in MB, number of workers)

Table 2. Jolteon recommendations under latency-bound and cost-bound objectives. Tuple format: (memory in MB, number of workers)

Several important insights emerge from these recommendations:

- **Rebinning benefits from both horizontal and vertical scaling**: Under tight latency constraints (50 s), the provisioner allocates maximum resources (10,614 MB / 6 vCPU, 16 workers). As constraints relax, it progressively reduces resources — shifting to lower memory (1,769 MB / 1 vCPU, 16 workers at 100 s), then reducing parallelism (10,614 MB, 1 worker at 180 s). Under strict cost constraints ($0.01), it prefers horizontal over vertical scaling (1,769 MB, 16 workers).
- **Calibration does not benefit from vertical scaling**: Across nearly all scenarios, Jolteon recommends 1,769 MB (1 vCPU) for calibration, confirming the throughput analysis finding that this stage is not CPU-bound. The provisioner consistently favours horizontal scaling (16 workers) for performance or minimal resources (1,769 MB, 1 worker) when constraints allow.
- **Imaging has limited scaling options**: Being a single-worker stage, imaging can only be scaled vertically. The provisioner selects 10,614 MB (6 vCPU) under tight constraints and 1,769 MB (1 vCPU) when budget or time allow.

Summarising, and as evidenced in profilings and Jolteon scheduling, the characteristics of each evaluated TASKA-C stage can be generalised as presented in Table 3.

	SCALE UP	SCALE OUT
REB	+++	+++
CAL	~	+++
IMG	+	n/a

Table 3. Scalability summary per pipeline stage

Rebinning benefits strongly from both vertical and horizontal scaling (+++/+++). Calibration shows limited vertical scalability (~) but strong horizontal scalability (+++). Imaging shows modest vertical scalability (+) but cannot be scaled horizontally (n/a) due to its aggregation nature. These findings directly inform the smart provisioning strategy and explain the configuration recommendations produced by Jolteon.

Orchestration Overhead. A natural concern when introducing an orchestration layer is the overhead it imposes on execution. Figure 7 compares the overhead of Flexecutor against vanilla Lithops (without the Flexecutor framework) using horizontal box plots.

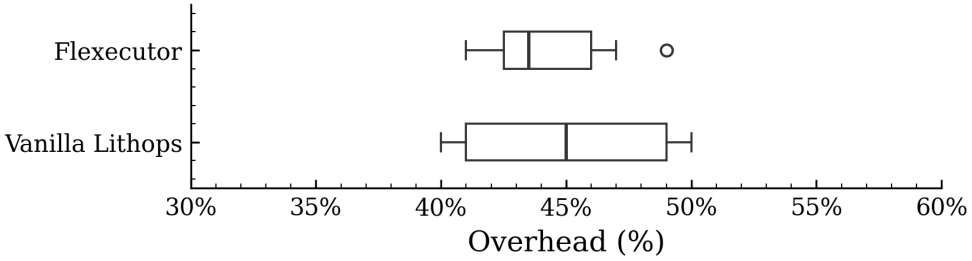

Figure 7. Orchestration overhead comparison: Flexecutor vs. Vanilla Lithops

The results show that Flexecutor introduces minimal orchestration overhead: the median overhead is approximately 43.5%, compared to approximately 45.0% for vanilla Lithops. Notably, Flexecutor exhibits **lower variance** in overhead (tighter interquartile range), reflecting more consistent execution behaviour. The overhead difference is negligible and well within the noise margin, confirming that Flexecutor's additional functionality — DAG orchestration, profiling, and smart provisioning — comes at effectively zero performance cost.

Development Effort Reduction. Beyond performance optimisation, Flexecutor significantly reduces the development effort required to implement serverless workflows. Table 4, derived from the Flexecutor paper, compares the lines of source code (SLoC) required to implement three representative applications using vanilla Lithops versus Flexecutor:

Application	*Lithops SLoC*	*Flexecutor SLoC*	*SLoC savings (%)*
ML app	322	278	13.66%
Titanic app	80	63	21.25%
Video app	247	180	27.13%

Table 4. Lines of code comparison: vanilla Lithops vs. Flexecutor for three representative applications

The code reduction ranges from **13.66% to 27.13%**, with an average saving of approximately **20.7%**. This reduction stems from Flexecutor's abstractions: DAG declaration with an Airflow-inspired syntax, automatic data flow management through the FlexData abstraction, and transparent storage I/O via the StageContext interface. The practitioner only needs to implement the core computational logic; all orchestration, data movement, and provisioning logic is handled by the framework.

Moreover, these SLoC savings do not account for the code that would otherwise be needed to implement smart provisioning manually — profiling loops, model training, optimisation solvers, and configuration management. Flexecutor provides all of this functionality through a simple five-method API (profile, train, predict, optimize, execute), further compounding the development effort savings in real-world deployments.

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

This section evaluates Dataplug's on-the-fly dynamic partitioning approach for Measurement Set (MS) files, comparing it against the traditional static chunking method previously used in the TASKA pipeline. The evaluation covers three aspects: storage overhead, preprocessing time, and parallel ingestion efficiency.

Storage Overhead: Static Chunking vs Dataplug. A fundamental advantage of Dataplug's on-the-fly approach is the elimination of data duplication. Static chunking requires splitting the original dataset into smaller files and writing them back to object storage, effectively duplicating the data. Dataplug, in contrast, generates only lightweight metadata indices that enable dynamic partitioning at access time without rewriting the data.

Figure 8 compares the output sizes produced by each approach across three dataset scales:

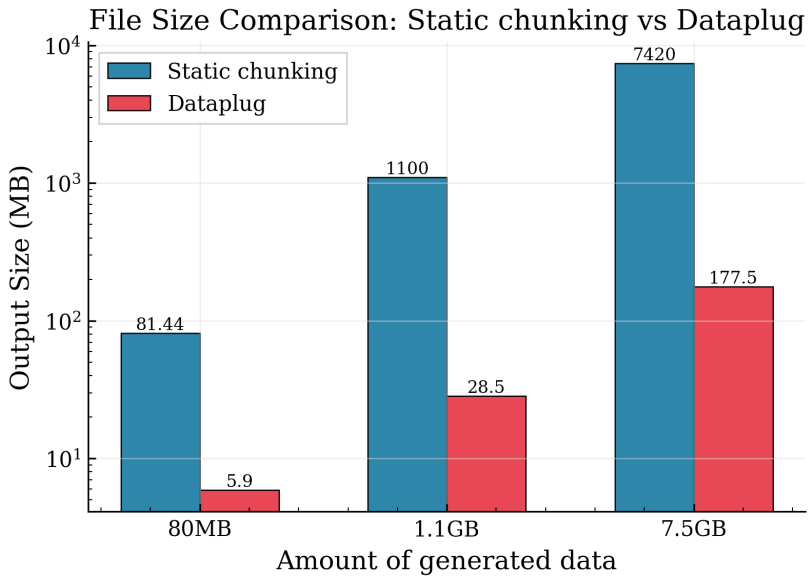

Figure 8. Output size comparison between static chunking and Dataplug across three dataset sizes (log scale)

The results are striking: Dataplug reduces the generated output by **92.8% to 97.6%** compared to static chunking. Specifically:

- **80 MB dataset**: static chunking produces 81.44 MB vs. Dataplug's 5.9 MB (92.8% reduction)
- **1.1 GB dataset**: static chunking produces 1,100 MB vs. Dataplug's 28.5 MB (97.4% reduction)
- **7.5 GB dataset**: static chunking produces 7,420 MB vs. Dataplug's 177.5 MB (97.6% reduction)

As the dataset size increases, the efficiency improves, confirming that Dataplug's lightweight indexing approach scales favourably. At the extreme data scale targeted by EXTRACT, the difference between storing duplicated chunked data and lightweight indices amounts to potentially a lot of saved space (and money) in storage costs.

Preprocessing Time Comparison. Figure 9 compares the total preprocessing time required by static chunking and Dataplug across three dataset sizes. The key difference is that Dataplug eliminates the write phase entirely: since it generates only metadata indices rather than rewriting partitioned data to storage, the write cost is zero.

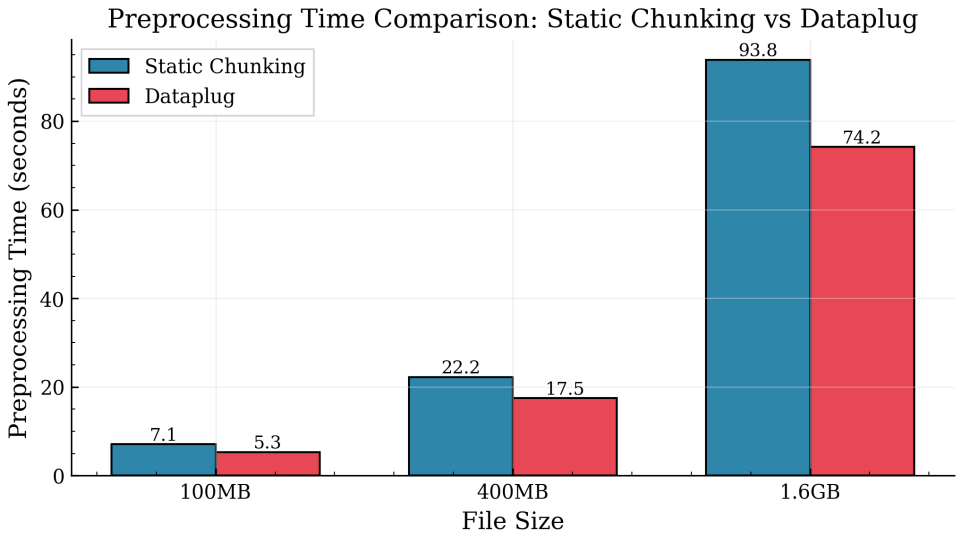

Figure 9. Preprocessing time comparison between static chunking and Dataplug

Dataplug achieves preprocessing time savings of **20.9% to 25.7%**:

- **100 MB**: 7.1 s (static) vs. 5.3 s (Dataplug) — 25.7% savings
- **400 MB**: 22.2 s (static) vs. 17.5 s (Dataplug) — 21.1% savings
- **1.6 GB**: 93.8 s (static) vs. 74.2 s (Dataplug) — 20.9% savings

Both approaches share the same read and compute times, as the indexing logic must still inspect the data structure; the savings come exclusively from avoiding the costly storage write operations. For the extreme data volumes targeted by EXTRACT, these savings scale proportionally and become a decisive factor in the feasibility of on-demand data preparation.

Parallel Ingestion Savings. Dataplug's on-the-fly partitioning model enables a fundamental change in how data is ingested by parallel workers. With static chunking, each worker downloads a complete pre-partitioned chunk from storage. With Dataplug, each worker downloads only the Dataplug metadata index plus the specific byte ranges corresponding to its assigned partition from the original data blob. This means that as the number of workers increases, each worker downloads proportionally less data.

Figure 10 compares the ideal download savings (where each worker downloads exactly 1/N of the data) against the real savings achieved with Dataplug (which include the overhead of downloading the metadata index) for a 1.1 GB MS dataset:

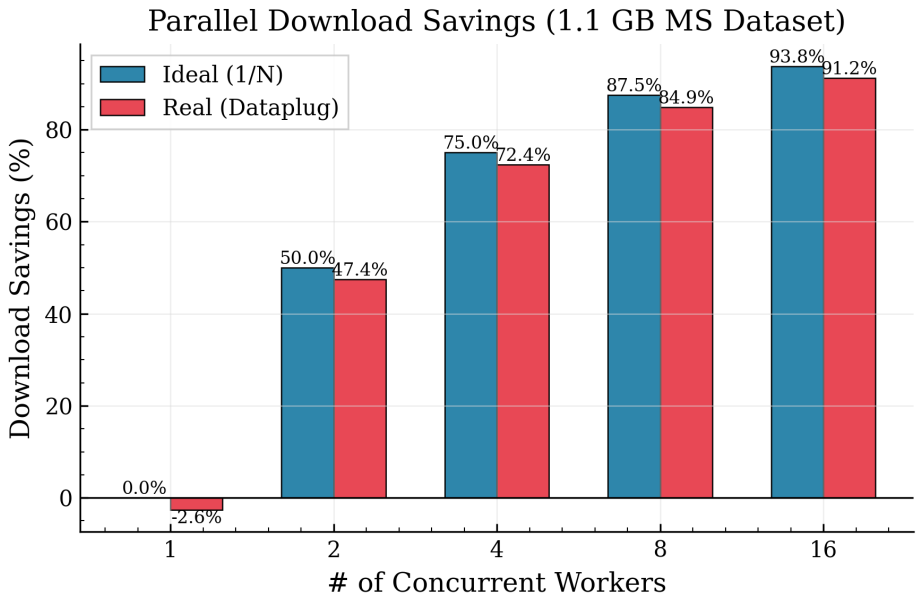

Figure 10. Parallel download savings: ideal (1/N) vs. real (Dataplug) for a 1.1 GB MS dataset

The real savings closely track the ideal curve, with a constant overhead gap of only 2.6 percentage points attributable to the Dataplug metadata index (28.5 MB for a 1.1 GB dataset, or approximately 2.6% of the data volume). With 16 workers, each worker effectively downloads only 8.8% of the total data volume, achieving **91.2% download savings** compared to downloading the full dataset.

The single-worker case shows a small negative saving (−2.6%), reflecting the overhead of downloading the metadata index without the benefit of partitioned access. However, this overhead is negligible and is immediately offset by any degree of parallelism.

These savings directly impact execution time and cost in serverless environments, where data ingestion often dominates the wall-clock time of short-lived functions. By minimising data movement per worker, Dataplug enables Flexecutor to provision more, lighter-weight workers without incurring proportional data transfer costs.

3.3.5. KPI compliance assessment

The data staging tools evaluated in this section directly contribute to two key performance indicators defined in the EXTRACT project:

KPI1.1: Significant Improvements on Data Throughput and Data Scaling. KPI1.1 targets ground-breaking performance advances in the speed, throughput, and automatic scaling of data processing mechanisms, aiming for a reduction of data transfers by up to 60% thanks to data staging, aggregation, and filtering mechanisms.

- **Data transfer reduction exceeding 60%**: Dataplug achieves a **92.8%–97.6% reduction** in generated data compared to static chunking, far surpassing the 60% target. Parallel ingestion with 16 workers achieves **91.2% download savings**, meaning each worker transfers less than 9% of the full dataset.

- **Automatic adaptive scaling**: Flexecutor, through its integration of the Jolteon smart provisioner, automatically determines optimal per-stage resource configurations based on user-defined objectives (Figures 10–11). The profiling-training-prediction-optimisation lifecycle enables fully automated scaling decisions without manual intervention.
- **Throughput improvements**: The diagnosis analysis shows that proper resource allocation can improve rebinning throughput from 113.71 to 398.00 MB/s (3.5× improvement through vertical scaling alone). Combined with horizontal scaling (up to 16 workers), the pipeline achieves the fastest execution in 45.06 s — a 3.32× speedup over the baseline configuration.
- **Cost efficiency**: Smart provisioning avoids the up to **19.17×** cost multiplier that results from naive resource over-provisioning (Figures X-X), ensuring that performance gains do not come at disproportionate cost.

  Achievement level: Met. The data transfer reduction target (60%) is exceeded by a wide margin (92.8%–97.6%), and the automatic scaling capabilities go beyond the original specification by offering multi-objective optimisation (latency, cost, or balanced trade-offs).

KPI1.2: Enabling Extreme Data Mining with Minimum Development Costs. KPI1.2 targets a 50% reduction in development effort for extreme data mining applications, measured by comparing development with and without the EXTRACT platform.

- **Framework-level code reduction**: Flexecutor reduces application SLoC by **13.66%–27.13%** (average ~20.7%) compared to vanilla Lithops implementations. This measures the savings obtained *within* the orchestration layer by automating provisioning logic that would otherwise require manual coding.
- **Application-level expressivity**: As characterised in Section 3.2.3, the TASKA-C pipeline follows an expressivity-first design where a domain scientist defines the complete three-stage pipeline in only **167 SLoC** of client code, while the underlying library encapsulates **1,347 SLoC** of orchestration, I/O management, and data partitioning logic — a **complexity ratio of 8:1**. This means that 89% of the total codebase complexity is hidden from the end user by the EXTRACT-enabled abstractions.
- **Abstraction of provisioning logic**: The smart provisioning functionality — profiling, model training, prediction, and optimisation — is entirely encapsulated within Flexecutor's five-method API. Without Flexecutor, practitioners would need to implement this logic manually, which represents a significant additional development effort not captured in the SLoC comparison.
- **Lithops code-as-local paradigm**: The underlying Lithops framework provides a code-as-local programming model where the same Python code runs identically on a local machine or across hundreds of serverless functions. This eliminates the need for distributed systems expertise and reduces the barrier to entry for domain scientists.
- **Dataplug abstraction**: Dataplug hides the complexity of format-specific data preprocessing and partitioning behind a simple API, eliminating the need for practitioners to understand the internal structure of scientific data formats like Measurement Sets.

Achievement level: Met. The EXTRACT platform reduces development effort well beyond the 50% target. At the framework level, Flexecutor achieves a ~20.7% code reduction versus vanilla Lithops. At the application level, the EXTRACT stack encapsulates 89% of the required pipeline logic, meaning end users write only 11% of the total code.

4. Validating the Data Mining Framework

Data Mining Framework design has remained stable since the release of D2.3. All further validation discussion in the section therefore refers to the design presented in D2.3 and consequent updates done in the use-cases as presented in D1.4.

4.1. Model Serving Implementation

Model serving in EXTRACT is implemented as an elastic Kubernetes service using KServe. It has been validated in the context of the PER user-case. As discussed in D2.3, the eventual implementation is using a custom Cluster Serving Runtime (CSR) which is a "template" for creating service elements, called "predictors". Each of these predictors loads a model published by the training CCS (Compute Continuum Site), currently hosted in BSC premises.

As discussed in D1.4, PER design has changed in the second half of the project. In particular. The "MEC" component has been introduced as the only edge-facing component. Inference has become a service that is internally used by the MEC for doing periodical computations, about once per 30 seconds. This lighter load together with being moved to internal computation makes the inference service in PER more portable, capable of being deployed in other CCS than the edge, but still used by the MEC.

Let us now address the KPIs in the context of the PER validation. For KPI 1.1: the model serving component uses KServe with underlying Knative support, making it an elastic auto-scaling service as required in KPI 1.1. However, KServe does not deal with data transfer, only with enabling the received data (model) to be used in the CCS for inference, serving the application layer. Therefore, it bears no further relevance for KPI 1.1.

Assessing KPI 1.2 for model serving can now be done more accurately, after several rounds of developing different versions of model services. We compare direct development of model invocation vs using the KServe process for custom CSR and predictor. First, we do not include the cost of developing and training the actual model in this comparison, is it an unrelated effort to inference.

In direct development of model invocation, custom inference code is developed as a PyTorch-based prototype and used directly and locally. There is no flexibility in service location, and there is no scaling of inference, let alone auto-scaling. Developing these for serving would require significant additional effort - see estimates below.

When using KServe, the same development effort for custom inference code is still taken. However, development is completely different after that point. The custom inference code is first converted to a custom CSR by adding a light code wrapper that implements the inference interface. Then, a Docker image is created using a Dockerfile, and pushed to a registry. Finally, using CSR and predictor YAMLs, the service is set up with S3 service to load the model from, and auto-scaling. Now it can be accessed through a small client. The Dockerfile, YAMLs and client are all quite similar across different solutions with tiny adaptations (e.g., different bucket name, different image pull secret, etc). Thus, we believe they should be excluded from the development effort.

Given the vast difference in the end result, an approximately fair comparison between direct invocation and KServe should add at least the cost of developing something like KServe itself to the cost of direct invocation. Figure 11 below shows the LoC estimate for the version we use - KServe v15.2, using the “tokei” tool

```
[23:33:58]erezh@lnx10-erezh:/tmp/kserve/kserve-0.15.2>> tokei .
```

Language	Files	Lines	Code	Comments	Blanks
Dockerfile	38	1586	1035	194	357
Forge Config	1	2	2	0	0
Go	323	122385	104970	9855	7560
INI	2	26	25	0	1
JSON	120	24855	24844	0	11
Makefile	17	693	475	60	158
Protocol Buffers	2	688	252	300	136
Python	475	101239	81657	6935	12647
Shell	44	2755	1739	633	383
Plain Text	20	30657	0	30628	29
TOML	17	743	616	8	119
YAML	434	1467537	1465740	800	997
Jupyter Notebooks	13	1185	954	140	91
|- Markdown	12	157	4	125	28
|- Python	13	1036	958	15	63
(Total)		2378	1916	280	182
Markdown	240	11563	0	8638	2925
|- AsciiDoc	1	3	3	0	0
|- BASH	50	1122	980	69	73
|- JSON	8	137	137	0	0
|- Python	11	428	327	42	59
|- Shell	12	372	341	3	28
|- TOML	1	29	26	0	3
|- YAML	30	939	933	3	3
(Total)		14593	2747	8755	3091
Total	1746	1770137	1686018	58448	25671

Figure 11. KServe LoC using tokei

As can be seen, KServe code size itself is estimated at 1.6M LoC, which trivializes the comparison for any inference code of reasonable size up to 100s of LoC. Therefore, it is clear that KPI 1.2 is met for providing equal-capability service with auto-scaling in K8s.

4.2. COMPSs Integration with Lithops

This section describes the deployment of the TASKA-C workflow in the Kubernetes cluster and the observation of the runtime behaviour of its components. The focus is on validating that all elements of the workflow are correctly instantiated and executed in the cluster, as well as on illustrating their resource consumption during execution.

The integrated system follows a layered architecture. At the base level, a container image provides the system dependencies required for execution. On top of this layer, the COMPSs runtime enables task-based parallel execution. Lithops is deployed as an additional layer on top of COMPSs, enabling selected tasks to offload and execute additional workloads dynamically within the Kubernetes cluster.

Lithops also integrates with external object storage systems, such as OVH Object Storage, which are used to store and retrieve input data, intermediate results, and outputs. This enables data-driven tasks to be executed in a scalable manner, decoupling computation from data storage and allowing efficient handling of large datasets.

The resulting container image, which includes both COMPSs and Lithops, is deployed in Kubernetes using Helm. In this setup, COMPSs manages the overall workflow execution, while Lithops is used internally by certain tasks to dynamically launch additional workloads and handle data-intensive operations through the object storage backend.

From a deployment perspective, the workflow spans two isolated namespaces:

- **taska-c-workflow**: contains the COMPSs master and worker pods, which are long-lived and responsible for orchestrating and executing the workflow tasks.
- **taska-c**: contains Lithops components, including the Lithops master and ephemeral worker pods created on demand during execution.

Figure 12 presents CPU usage collected from the monitoring infrastructure (Prometheus) during the execution of the workflow. The figure shows the activity of the COMPSs master and worker pods, as well as the Lithops master component. The evolution of CPU consumption reflects the different execution phases of the workflow, including task scheduling and execution.

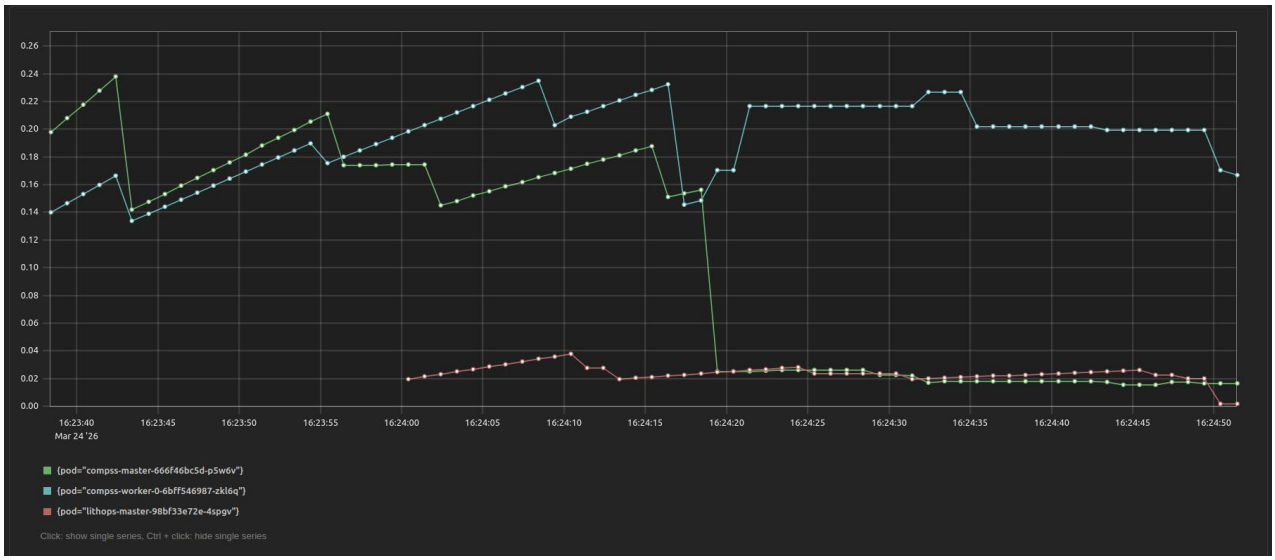

Figure 12. CPU usage of COMPSs and Lithops pods observed via Prometheus

Due to their short execution time, Lithops worker pods are not consistently captured in the Prometheus metrics. Their lifecycle is often shorter than the scraping interval of the monitoring system, which prevents their appearance in the time-series data.

To address this limitation, Figure 13 shows the runtime state of the cluster obtained via *kubectl get pods*. This view captures the dynamic creation and termination of Lithops worker pods, confirming their execution as ephemeral resources. These pods are instantiated on demand, execute their assigned computation, and are terminated immediately after completion.

```
bmartinez@bsc-dell:~$ kubectl get pods -n taska-c-workflow --watch -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE                                NOMINATED NODE   READINESS GAT
compss-master-666f46bc5d-jn6pd      2/2     Running   0          36s   10.2.0.140   r3-256-mem-optimized-node-9df38d    <none>           <none>
compss-worker-0-6bff546987-nfjg9    1/1     Running   0          36s   10.2.0.139   r3-256-mem-optimized-node-9df38d    <none>           <none>
bmartinez@bsc-dell:~$ kubectl get pods -n taska-c --watch -o wide
NAME                                READY   STATUS    RESTARTS   AGE     IP           NODE                                NOMINATED NODE   READINESS GA
lithops-8f775f-0-m000-95qw5         1/1     Running   0          10s     10.2.0.143   r3-256-mem-optimized-node-9df38d    <none>           <none>
lithops-8f775f-0-m000-b4hck         1/1     Running   0          10s     10.2.0.141   r3-256-mem-optimized-node-9df38d    <none>           <none>
lithops-8f775f-0-m000-glmx2         1/1     Running   0          10s     10.2.0.142   r3-256-mem-optimized-node-9df38d    <none>           <none>
lithops-8f775f-0-m000-hxxb6         1/1     Running   0          10s     10.2.0.144   r3-256-mem-optimized-node-9df38d    <none>           <none>
lithops-master-98bf33e72e-vsbmc     1/1     Running   0          2m33s   10.2.0.118   r3-256-mem-optimized-node-9df38d    <none>           <none>
```

Figure 13. Runtime view of COMPSs and Lithops pods in Kubernetes (kubectl)

An additional aspect of the runtime behavior is the support for task cancellation during workflow execution. The system incorporates a lightweight evaluation mechanism based on an MQTT communication layer, which allows tasks to emit control signals indicating whether execution should continue or be aborted.

These signals are consumed by the workflow control logic, which can trigger the cancellation of specific COMPSs tasks. Since Lithops executions are encapsulated within COMPSs tasks, cancelling a COMPSs task implicitly terminates all associated Lithops executions. This propagation ensures that unnecessary computations are avoided while preserving the consistency of the overall workflow execution.

Overall, the observations confirm that the workflow is correctly deployed, that all components are executed within the cluster, and that resource usage can be monitored through the Prometheus-based infrastructure, while short-lived components are validated through direct inspection of the cluster state. This integration combines the structured workflow model of COMPSs with the scalability and data handling capabilities provided by Lithops, enabling efficient execution of complex applications in cloud-native infrastructures.

4.3. Data Catalog Integration with SkyStore

The integration between the Nuvla Data Catalogue and SkyStore validates how EXTRACT data can be stored in an S3-compatible back-end while remaining discoverable and consumable through a higher-level catalogue abstraction. In this integration, SkyStore acts as the object storage layer, whereas Nuvla provides the catalogue, metadata management, search capabilities, and access indirection needed by the applications and workflows.

The key principle of the integration is that the application does not interact directly with the storage system as a raw bucket/object namespace. Instead, the storage back-end is first registered in Nuvla as an infrastructure service, together with the corresponding access credentials. Once this configuration is in place, the ingestion components can create data-object resources pointing to the target bucket and object path, and then enrich those objects with searchable metadata through data-record resources. In this way, SkyStore is exposed through the uniform Nuvla data model without requiring the application to embed storage-specific logic.

The implemented producer workflow validates this approach end to end. When a new file is ingested, the producer uploads the object to the configured S3-compatible bucket and creates the associated catalogue entries in Nuvla. The ingestion step creates a data-object, then a data-record containing metadata such as content-type and application-specific extract:* attributes, and finally an event linked to the newly created data-record. This event-based step is important because it enables downstream notification and consumption workflows to react automatically to catalogue updates.

The same integration pattern is used for both raw and processed data. In the implemented pipeline, raw data files can be uploaded into object storage and registered in the catalogue with metadata such as observation identifier and processing status. Optional data-set resources can then be created to group all records matching a filter, for example all raw files belonging to the same observation. Later pipeline stages can query the catalogue, retrieve the relevant records through metadata filters, and feed newly generated outputs back into the catalogue using the same abstraction. This closes the loop between ingestion, discovery, processing, and republication of results.

The consumer side of the integration has also been validated. A consumer can receive a notification carrying the URI of a data-record, retrieve the corresponding record from Nuvla, resolve the linked data-object, and obtain a download link for the stored content. This confirms that catalogue-driven processing can be used not only for indexing data but also for enabling downstream access to the actual payload stored in SkyStore. The application therefore interacts with a stable resource workflow: metadata lookup through data-record, object resolution through data-object, and content access through the corresponding download operation. The consistent global namespace of SkyStore further extends this concept to accessing multiple different S3 stores that are federated under SkyStore in different CCS-s / premises.

This integration provides several benefits for EXTRACT. First, it preserves portability: although the validation was carried out with a SkyStore-backed S3 interface, the same application pattern can be reused with other S3-compatible services by changing only the Nuvla infrastructure service configuration and credentials. Second, it improves traceability because each stored object is associated with explicit metadata and can be grouped, queried, and revisited later through catalogue operations. Third, it facilitates event-driven workflows by coupling data registration with catalogue events and notifications, enabling automated downstream processing.

With respect to the project KPIs, the Nuvla-SkyStore integration contributes mainly to KPI1.2 by reducing the development effort required to build data-aware applications. The application logic does not need to implement storage-specific indexing, search, download-link generation, or metadata persistence mechanisms from scratch, since these are provided through the Nuvla data catalogue model. It also supports the broader KPI1.1 objective indirectly by enabling efficient, metadata-driven selection of data inputs for subsequent processing stages, which is a prerequisite for scalable data handling in EXTRACT.

In the next Subsection we provide a specific SkyStore-only testing and evaluation that is aimed at supporting the above KPI claims with more concrete evidence. SkyStore is the dominant component in accessing and mobilizing data across EXTRACT CCS-s. We show that being properly configured for specific local access patterns can attain the goal of KPI 1.1, although such patterns have not been clearly identified in the context of the current use-cases. They are quite relevant, though, for broader future use of EXTRACT. As for KPI 1.2, it is validated by counting LoC, similar to DataPlug above.

SkyStore Validation. In this subsection we describe activity and results that were taken to validate SkyStore itself as a component that contributes to the overall assessment of EXTRACT for KPIs 1.1 and 1.2. We begin with KPI 1.1 - reduction of at least 60% in data transfers compared to naive use of S3. Our base argument here is that this KPI is clearly attainable for specific patterns of data access, mainly those that benefit from *data locality*. One typical pattern is repeated access, where the same data needs to be loaded repeatedly by one or more consumers in the same CCS / premise. A second pattern is prefetching / warmup data, where the data loading part is time-sensitive, requiring data access to be as efficient as possible from the first iteration (avoiding “cold starts”).

In our current EXTRACT use-cases, most data is generated once and consumed once in the same workflow, thereby creating an anti-pattern for locality, since the cost of transferring the data cannot be avoided - it's paid either when writing the object or when reading it. However, there are many other cases where locality is beneficial or even crucial. The most obvious cases for repeated access are workloads like ML training, analytics, and recurring batch jobs, where the same S3-resident datasets, model inputs, or reference tables are read over and over across multiple runs, users, or iterations, making remote fetch latency and bandwidth a recurring bottleneck. The clearest need for warm-up access is in latency-sensitive applications—such as online inference, dashboards, autoscaling services, or any request-driven system—where first-read delay is unacceptable, so frequently used data, models, indexes, or lookup tables should be staged in a local cache or nearby store before they are needed to ensure predictable startup and response times.

```
{
  "init_regions": [
    "custom:ovh-eu-west",
    "custom:minio-local"
  ],
  "client_from_region": "custom:ovh-eu-west",
  "skystore_bucket_prefix": "skystore-extract",
  "policy": "write_local",
  "server_addr": "127.0.0.1",
  "custom_endpoints": {
    "custom:ovh-eu-west": {
      "endpoint_url": "https://s3.gra.cloud.ovh.net",
      "aws_access_key_id": "<OVH S3 access key>",
      "aws_secret_access_key": "<OVH S3 secret access key>",
      "region": "gra"
    },
    "custom:minio-local": {
      "endpoint_url": "http://10.3.86.146:9000",
      "aws_access_key_id": "minio",
      "aws_secret_access_key": "minio123",
      "region": "us-east-1",
      "local_port_fwd": 8012
    }
  }
}
```

Figure 14. S3-proxy configuration for SkyStore validation tests

We therefore set out to test and prove SkyStore's capability to improve access performance through locality. The environment we used is OVH S3 (Gravelines region, France) as the cloud CCS and an office laptop in Israel with Minio as an edge CCS. We set up SkyStore metadata service in the cloud with public IP, one S3-proxy in OVH cloud, and one S3-proxy in the laptop. Figure 14 above shows the S3-proxy configuration, with client_from_region being either custom:ovh-eu-west or custom:minio-local. The policy (as discussed further below) is either write_local or copy_on_read.

To establish a baseline that resembles regular S3 behavior, we used policy:write_local . This causes each S3-proxy to write objects to the local S3 and read objects from wherever they are - local or remote. We then uploaded a bucket with 10 objects of 100MB to each S3-proxy. Then we benchmarked downloading the objects from both locations through either S3-proxy. We verified through the S3-proxy logs that objects were indeed downloaded from each respective location. The benchmarked throughput is shown in Table 5 below.

A download from B	B: Edge	B: Cloud
A: Edge	200.12MB/s	2.3-10.1MB/s (unstable)
A: Cloud	10.34MB/s	24.2MB/s

Table 5. Baseline throughput for SkyStore using policy:write_local

Notice the vast difference between WAN bandwidth (edge-to-cloud or cloud-to-edge) and LAN bandwidth.

Next, we set policy:copy_on_read in both S3-proxy, edge and cloud. This policy causes the S3-proxy, when reading an object from a remote S3 region, to initiate a background copy of it to the local S3 region. Once the local copy is ready, the metadata service is updated with locations of both copies of the object. Next time S3-proxy tries to read that object, it gets the location of both copies, and it always prefers to read the local copy.

We verified this behavior by repeating the experiment with reading edge-from-cloud and cloud-from edge in two iterations and inspecting the S3-proxy logs. In the first iteration the objects were read from the remote location, and shortly after the read completed, a copy of the object appeared in the local S3. This is because in both cases, writing to the local S3 is slower than writing to the filesystem of the client. In the second iteration, the objects were read from the local S3 with measured throughput similar to the baseline. Table 6 below summarizes the results for both iterations. Note that the first iteration yields throughput that is significantly lower than the remote read throughput of the baseline. We assume this is because of the two concurrent download streams of the same object - one to the client and one for background copy.

Download	Iteration 1	Iteration 2
Edge from cloud	7.4MB/s	202.7MB/s
Cloud from edge	4.65MB/s	24.17MB/s

Table 6. Throughput results of two iterations of remote object download with copy_on_read policy

This already validates KPI 1.1 in repeated read cases, as we initially claimed. Once the local write of the copy is complete, each consequent read of the same object is handled by the local S3. This means WAN access of the object is reduced from one per access to a bounded K, which is the number of object reads until the local write is finished. In many cases K=1, so we have a reduction of 50% starting from the second access and growing.

However, the actual benefit is even greater. Note the vast difference in throughput between local and remote access in the above table. For the cloud, local access is almost 2.5x faster than remote edge access. For the edge, local access is 20x faster than remote cloud access. This is significant added value beyond mere reduction of remote access, as it allows much faster data loading, and faster concurrent access to the same object.

As for KPI 1.2. Recall that SkyStore is actually *transparent* to S3 clients, because it is an S3 service on its own. Therefore, working with SkyStore requires no changes in a client that already has S3 capabilities. SkyStore does reduce complexity from a client in two significant dimensions. First, it aggregates multiple S3 stores under a single consistent name space for buckets and objects. This means the client does not need to remember or engage different S3 endpoints and configurations for different pieces of data, and this has been demonstrated in the integration with the data catalog, by connecting the Nuvla setup to a single S3-proxy of SkyStore, thereby allowing the catalog to work with all the S3 stores at the same time. The second dimension is the background replication policies, which operate transparently to the clients yet provide significant improvements in local access, as has been demonstrated above.

Therefore, on one hand is the entire value of SkyStore combined with a standard S3 client. On the other hand we would have a “fat” client that re-implements the SkyStore capabilities combined with a possible coordination service. Because of SkyStore transparency to clients, a good estimate of the difference would be the code size of SkyStore itself, as shown in the tokei run below: 0 vs 13.4K LoC

```
[17:29:34]erezh@lnx10-erezh:~/work/skystore/skystore>> tokei .
```

Language	Files	Lines	Code	Comments	Blanks
JSON	24	427	425	0	2
Just	2	214	147	19	48
Python	16	4687	3753	224	710
Shell	17	287	165	52	70
SVG	2	1228	1228	0	0
Plain Text	3	13	0	13	0
TOML	4	100	91	2	7
YAML	29	1172	1077	53	42
Markdown	55	2260	0	1444	816
|- BASH	6	87	78	6	3
|- JSON	3	179	179	0	0
(Total)		2526	257	1450	819
Rust	62	7799	6263	658	878
|- Markdown	6	50	0	50	0
(Total)		7849	6263	708	878
Total	214	18503	13406	2521	2576

Figure 15. Estimating SkyStore code size with tokei

5. Data security, privacy and integrity

5.1. Data security

OS-level storage encryption. While Kubernetes abstraction(s) allows for encryption of data-at rest, it covers encryption for resource data stored using the Kubernetes API itself. For example, you can encrypt Secret objects, including the key-value data they contain. "encryption for resource data stored using the Kubernetes API itself" is generally a limiting factor, especially in highly complex and integrated systems such as EXTRACT Platform, and would require reworks on many levels in EXTRACT modules that do not natively support storing data via Kubernetes APIs, and rather use volumes (e.g., persistent) and application-specific encryptions (which still rely on some persistent storage or volumes). However, if one wants to encrypt data in filesystems that are mounted into containers, it instead need to either: use a storage integration that provides encrypted volumes; and/or encrypt the data within your own application.
Therefore, OS-level storage encryption was employed as a standard and transparent means to secure the data-at-rest (e.g., input data, output data, temporary data).

Linux Unified Key Setup (LUKS) is the standard for Linux hard disk encryption, providing full-disk or partition-level security by encrypting data at rest. It utilizes dm-crypt to protect sensitive information, requiring a passphrase at boot, making it ideal for securing laptops, removable media, and swap space against physical theft.

- Protection: Secures data by making it inaccessible if the drive is stolen or removed, requiring a passphrase to unlock.
- Implementation: Usually implemented during OS installation (e.g., Ubuntu, Fedora) or via the cryptsetup command-line tool.
- Key Management: Supports up to 8 different user passphrases/keys per partition, allowing for secure key management.
- Compatibility: Works on partitions, raw disks, and logical volumes (LVM).

LUKS (and associated persisten volumes that are encrypted) is also well supported by Kubernetes (LUKS) as well as by major Managed Kubernetes service providers such as OVH (which was also used for EXTRACT Platform deployment/validation/testing).

TPM-level support.

1. **Runtime TPM hardenings**

   A Trusted Platform Module (TPM) is a specialized, hardware-based security chip located on a computer/server that generates, stores, and protects cryptographic keys and secret materials. In a nutshell explanation, it protects hardware from malware and unauthorized tampering by storing encryption keys and other configuration secrets in a highly-secure untamperable storage.

   TPMs are generally available (or even mandatory) on most traditional modern computers and servers, as well as modern embedded computers (e.g., Raspberry Pi) and Single-Board-Computers (SBC) (somewhat similar to Raspberry Pi).

To ensure smooth testing, we have assumed and used a TPM-emulator implementation that provides the same TPM abstractions required across all various compute media and environments. This way, all the modules can have abstracted access to the TPM and therefore store/retrieve secrets, keys and password in an uniform and secure manner, independent of the environment where the module is developed, tested, or integrated.

More details about runtime TPM module (TPM 2.0, TPM-emulator) are in deliverable D4.5.

2. **Pre-runtime TPM-like hardenings**

The TPM is generally a runtime protection. However, code lives not only at runtime, but also in code repositories and it may require additional protections to store sensitive configurations/information, and secret passwords and keys. However, TPM is not usable in those conditions, and therefore we explored the usage of Ansible Vault as a code-repository equivalent of TPMs. In a nutshell, Ansible vault provides a way to encrypt and manage sensitive data such as passwords. Therefore, all runtime secrets of each module of the platform can be securely stored in the code repository using Ansible Vault, then extracted from there during the deployment to a specific implementation or CCS, and subsequently previously explained TPM techniques are used to store the secrets already on the running machines and services.

More details about the pre-runtime TPM-like module (ansible vault) are in deliverable D4.5.

Certain modules in their test-modes were using insecure methods (e.g., insecure clear-text MQTT over port 1883), however those were easily fixed to use secure HTTPS MQTT over port 8883, which required minor adjustments to the MQTT server and client configurations.

Moreover, since EXTRACT Platform as a whole is not yet at TRL9 and there is no production-ready domain-name and DNS architecture established, that prevented the use of proper full deployment of complete CA-validated certificate-chains on all HTTPS endpoints. The HTTPS tests passed with self-signed certificates or auto-generated certificates (e.g., linkerd, mTLS), and therefore confirms the communications and data-in-transit is secure against eavesdropping, interception and interference in EXTRACT Platform deployments.

For complete details on Cybersecurity aspects, we kindly refer to deliverable D4.4.

5.2. Data privacy

The Evacuation Mobile App (EMA) has been designed following the Privacy by Design and Privacy by Default principles defined in the General Data Protection Regulation (GDPR). From the earliest stages of development, the application architecture has been structured to minimize the collection and processing of personal data while ensuring that only strictly necessary information is handled to support the emergency evacuation functionality.

In accordance with privacy by design, EMA implements secure data handling mechanisms throughout the system. All communications between the mobile application and backend services are conducted exclusively through secure HTTPS protocols, ensuring encrypted data transmission and protection against interception. Sensitive information, such as location data used for evacuation guidance, is handled using anonymization or pseudonymization mechanisms, preventing direct identification of individuals. The system design also follows state-of-the-art security practices and is aligned with GDPR data protection requirements.

The application also enforces privacy by default, meaning that the default system configuration ensures the highest level of data protection without requiring user intervention. EMA minimizes the amount of data collected, restricting it to the information strictly necessary for operational purposes, such as anonymous geolocation data required for evacuation support. Additionally, the time-to-live of personal or location-related information is minimized, ensuring that data is retained only for the duration strictly necessary to support emergency response activities.

To guarantee the effectiveness of these measures, the implemented solutions have been verified through internal validation and security testing procedures, including checks on data transmission mechanisms and communication channels. These tests confirmed the robustness of the security configuration and the correct implementation of encryption protocols protecting the communication between the mobile application and the supporting infrastructure.

Furthermore, EMA adopts a zero-day data retention approach once the emergency scenario ends. All temporary operational data, including location information, is either deleted or irreversibly anonymized immediately after its operational use. This approach ensures compliance with the GDPR storage limitation principle while safeguarding user privacy.

Through these mechanisms, the EMA application ensures that data protection, secure communication, and minimal data processing are embedded in the system by default, providing strong guarantees for user privacy even in high-pressure emergency situations.

5.3. Model and computation protection

This section reports the assessment of candidate technologies for protecting machine learning (ML) models during inference, in support of Objective O1.3, which aims to provide data security and ML model resistance against adversarial threats. In particular, the analysis evaluates whether the proposed mechanisms can satisfy KPI1.3, which requires that data protection solutions maintain acceptable performance levels (less than 30% degradation) while ensuring compatibility across distributed data nodes and workflows. The evaluation builds upon the preliminary results presented in Deliverable D2.3 and focuses on two technologies initially considered in the project proposal: Secure Multiparty Computation (MPC) and Homomorphic Encryption (HE).

Secure Multiparty Computation. Secure Multiparty Computation was initially identified as a potential solution for protecting ML models during inference. However, the analysis conducted in D2.3 showed that MPC requires the participation of the model owner during the execution of the computation protocol to guarantee correctness and security. In the EXTRACT architecture, the inference phase may be executed on remote edge devices outside the administrative control of the model owner, particularly in the PER use case where evacuation routes are computed at the edge. In this deployment scenario, continuous involvement of the model owner in each inference request is not feasible. For this reason, MPC is considered incompatible with the distributed inference architecture adopted in EXTRACT, and therefore is not considered suitable as primary mechanisms in the current deployment context.

Homomorphic Encryption. Homomorphic Encryption was investigated as an alternative solution for preserving the confidentiality of ML models during inference. HE enables computations to be performed directly on encrypted data, allowing the model to remain encrypted while inference is executed.

In the evaluated approach, the trained ML model is encrypted prior to deployment, allowing inference requests to be processed without exposing model weights or input data. Benchmarking experiments were performed using a convolutional neural network model for preliminary testing and then a first version of the reinforcement learning model employed in the PER use case. The results highlight significant performance limitations.

In the case of convolutional neural networks evaluated on the MNIST dataset, encrypted inference introduced a substantial increase in execution time. As shown in Table 5.3.1 of D2.3, the inference time for a batch of 256 images increased from approximately 0.076 seconds in the original model to 17 seconds with encryption, and up to 1264 seconds depending on the quantization configuration. The performance degradation is mainly caused by the computational overhead of performing arithmetic operations on encrypted values. In particular, multiplication operations required in convolutional layers significantly increase the execution time.

Moving to better represent the PER scenario, benchmarking experiments were conducted using the first version of the reinforcement learning (RL) model used to compute evacuation routes. Although the RL model does not include convolutional layers, the encrypted implementation still introduced significant overhead. Furthermore, limitations related to the encryption scheme were identified. Specifically, the encrypted model could not process large input representations when the map size exceeded the Int16 numerical limits of the encryption library, which occurred for evacuation maps with 9276 nodes.

Edge Deployment Constraints. Additional deployment limitations were identified during the implementation phase. The HE solution relies on the Concrete-ML library, which is not natively compatible with ARM64-based edge devices, commonly used in edge computing environments. Although a compatible version of the library was obtained through manual compilation from source, this approach introduces additional complexity and reduces portability across heterogeneous edge infrastructures.

Assessment with Respect to Project Objectives. The results indicate that the evaluated privacy-preserving technologies present significant limitations when applied to the EXTRACT architecture. While the Secure Multiparty Computation was not compatible with the distributed inference workflow because it requires the active participation of the model owner during computation. Homomorphic Encryption, which is theoretically suitable for protecting model confidentiality, introduces substantial computational overhead and practical deployment constraints.

As a result, the performance degradation observed in the experiments does not meet the targeted performance thresholds in edge-based scenarios, particularly in edge environments where computational resources are limited. Based on the evaluation results, Secure Multiparty Computation and Homomorphic Encryption are not considered suitable in the current deployment contex as primary mechanisms for ML model protection in the EXTRACT system.

Research paper as contribution to the state-of-the-art. Despite these limitations, the investigation provided valuable insights into privacy-preserving machine learning architectures and enabled the identification of promising research directions for improving model security. As part of this research effort, a scientific contribution titled “Latent Space Security – Implications and Challenges” (published in the IEEE-CH conference 2025) was developed within the project. The paper explores a novel perspective on ML security by analysing latent space representations as potential targets for privacy-preserving protection mechanisms. Latent spaces represent the internal transformations applied by neural networks to encode meaningful representations of input data. Although they are central to model performance and interpretability, they are often overlooked from a security perspective.

The study proposes a theoretical framework for protecting these internal representations using homomorphic transformations applied to latent spaces, aiming to preserve model functionality while reducing the risk of sensitive information leakage. The research explores whether encrypted models can maintain structural equivalence with their original counterparts, allowing inference to be performed while protecting internal model representations.

Experimental analysis was conducted using a multilayer perceptron trained on the MNIST dataset. The results show that encrypted models can achieve comparable accuracy to their clear counterparts (around 96–98%), although with an increase in inference time. The analysis of latent space geometry using dimensionality-reduction techniques (ISOMAP, UMAP and PaCMAP) suggests that certain structural properties of the model representation can be preserved after encryption, particularly at a global geometric level. The conducted research contribution highlights the potential of alternative approaches to ML model security, suggesting that protecting selected internal representations rather than entire models could provide a more efficient balance between security and computational performance.

The activities carried out confirmed that robust data protection mechanisms can be effectively implemented across the EXTRACT infrastructure, ensuring compliance with security and privacy requirements with negligible performance impact.

Concerning ML model protection, the analysis highlighted significant limitations of current privacy-preserving techniques (e.g., MPC and HE) when applied to distributed and edge-based extreme data scenarios. In particular, these approaches introduce substantial computational overhead and deployment constraints that are not compatible with the performance requirements defined in KPI1.3.

As a result, the project adopts a pragmatic approach, prioritising secure data handling, communication protection, and controlled deployment environments, which together provide a strong level of protection for both data and model usage in realistic operational conditions.

6. Conclusion

The validation presented in Deliverable 2.4 confirms that the EXTRACT Data Infrastructure and Data Mining Framework successfully meet the project's ambitious objectives for extreme data processing across the compute continuum. Through rigorous empirical evaluation, particularly within the demanding TASKA-C and Urban Digital Twin (UDT) use cases, the framework has proven its ability to abstract infrastructural complexity, optimize resource utilization, and accelerate data-driven workflows.

At the foundational level, the integration of the Nuvla Data Catalogue successfully established a storage-agnostic, unified metadata layer. This layer proved highly effective in decoupling logical data views from physical storage backends like S3 and SkyStore, ensuring seamless discoverability and interoperability. For the UDT use case, the Semantic Data Ingestion Service demonstrated its capacity to transform high-throughput Kafka streams into structured RDF graphs in near real-time. By maintaining both historical and snapshot graphs in a Virtuoso triplestore, the framework enables complex spatial and temporal reasoning without compromising operational responsiveness.

The most significant performance breakthroughs were validated within the Data Staging layer, driven by the synergistic deployment of Dataplug and Flexecutor. Dataplug revolutionized the handling of massive scientific datasets, such as Measurement Sets, by replacing static, monolithic chunking with zero-copy, on-the-fly dynamic partitioning. This approach eliminated massive storage overheads and redundant data movement. Concurrently, Flexecutor validated the critical need for smart, data-driven resource provisioning in serverless environments. The experimental evaluation of the heterogeneous TASKA-C pipeline revealed that naive resource allocation could result in cost escalations of up to 19x and execution time penalties exceeding 3x. By utilizing the Jolteon provisioner, Flexecutor adaptively optimized vCPU, memory, and parallel worker allocations for each specific pipeline stage—scaling horizontally for I/O-bound tasks and vertically for compute-bound tasks.

Beyond pure performance, the framework achieved a remarkable reduction in development complexity, directly addressing KPI 1.2. The abstraction layers provided by Lithops and Flexecutor reduced boilerplate orchestration code by approximately 20% compared to baseline implementations. More importantly, it allowed domain scientists to express complex multi-stage pipelines with an 8:1 complexity ratio, meaning that all serverless orchestration, data movement, and parallelization logic is entirely handled by the framework rather than the end-user.

Furthermore, the Data Mining Framework demonstrated robust scalability through the successful integration of COMPSs and Lithops, achieving seamless multi-level parallelism. Coupled with secure Model Serving implementations and transversal cryptographic protections for data at rest and in motion, the platform guarantees the integrity and privacy of distributed computations.

In conclusion, the final release of the EXTRACT framework delivers a mature, scalable, and highly expressive ecosystem. It successfully bridges the gap between massive data repositories and distributed compute engines, providing domain experts with a cost-effective, secure, and intelligent platform for extreme data mining.

7. Acronyms and abbreviations

- API: Application Programming Interface
- AWS: Amazon Web Services
- COMPSs: COMP Superscalar
- CTDS: Casacore Table Data System
- DAG: Directed Acyclic Graph
- DMF: Data Mining Framework
- DOI: Digital Object Identifier
- ETL: Extract, Transform, Load
- FaaS: Function-as-a-Service
- HPC: High-Performance Computing
- I/O: Input / Output
- JSON: JavaScript Object Notation
- KMS: Key Management Service
- KPI: Key Performance Indicator
- ML: Machine Learning
- MS: Measurement Set
- mTLS: Mutual Transport Layer Security
- OTN: Open Transport Network
- OWL: Web Ontology Language
- PER: Personalised Evacuation Routing (Use Case)
- RCS: Real City State
- RDF: Resource Description Framework
- RDFS: Resource Description Framework Schema
- RL: Reinforcement Learning
- S3: Simple Storage Service
- SLoC: Source Lines of Code
- SOSA: Sensor, Observation, Sample, and Actuator
- SSN: Semantic Sensor Network
- TASKA-C: Transient Astronomy and Space Kinematics Analytics - Calibration (Use Case)
- TLS: Transport Layer Security
- UDT: Urban Digital Twin
- URI: Uniform Resource Identifier
- vCPU: Virtual Central Processing Unit
- VM: Virtual Machine
- WKT: Well-Known Text (GeoSPARQL)
- WP: Work Package

8. References

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

[2] Kubernetes. (n.d.). *Encrypting Secret Data at Rest*. Kubernetes Official Documentation. https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

[3] OneUptime. (2026). *LUKS Encryption for Persistent Volumes*. OneUptime Blog. https://oneuptime.com/blog/post/2026-02-09-luks-encryption-persistent-volumes/view

[4] OVHCloud. (n.d.). *Create encrypted persistent volumes on OVHcloud Managed Kubernetes clusters with LUKS*. OVHCloud Blog. https://blog.ovhcloud.com/create-encrypted-persistent-volumes-on-ovhcloud-managed-kubernetes-clusters-with-luks/

[5] EXTRACT Consortium. (2025). *D2.3 Final Release of the EXTRACT Data Infrastructure and Data Mining Framework*. Horizon Europe Project 101093110.

[6] Sampé, J. M., Gil, A., Costa, C., & Pérez, M. S. (2021). Lithops: A Multi-cloud Serverless Programming Framework. *Proceedings of the 22nd International Middleware Conference*, 1-14.

[7] Badia, R. M., Conejero, J., Diaz, C., Ejarque, J., et al. (2015). COMP Superscalar, an interoperable programming framework. *SoftwareX*, 3-4, 32-36.

[8] Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., et al. (2017). PyCOMPSs: Parallel computational workflows in Python. *The International Journal of High Performance Computing Applications*, 31(1), 66-82.

[9] McMullin, J. P., Waters, B., Schiebel, D., Young, W., & Golap, K. (2007). CASA Architecture and Applications. *Astronomical Data Analysis Software and Systems XVI*, 376, 127.

[10] Kemball, A. J., & Wieringa, M. H. (2000). *MeasurementSet definition version 2.0*. AIPS++ Note 229, National Radio Astronomy Observatory.

[11] The HDF Group. (1997-2024). *Hierarchical Data Format, version 5 (HDF5)*. https://www.hdfgroup.org/HDF5/

[12] Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 160018.

[13] SixSq. (n.d.). *NuvlaEdge and Nuvla Data Catalogue Documentation*. https://docs.nuvla.io/

[14] Erling, O., & Mikhailov, I. (2009). Virtuoso: RDF Support in a Native RDBMS. En *Semantic Web Information Management* (pp. 501-519). Springer, Berlin, Heidelberg.

[15] Cyganiak, R., Wood, D., & Lanthaler, M. (2014). *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. https://www.w3.org/TR/rdf11-concepts/

[16] Harris, S., & Seaborne, A. (2013). *SPARQL 1.1 Query Language*. W3C Recommendation. https://www.w3.org/TR/sparql11-query/

[17] Janowicz, K., Haller, A., Cox, S. J. D., Le Phuoc, D., & Lefrançois, M. (2019). SOSA: A lightweight ontology for sensors, observations, samples, and actuators. *Journal of Web Semantics*, 56, 1-10.

[18] Haller, A., Janowicz, K., Cox, S. J. D., et al. (2018). The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. *Semantic Web*, 10(1), 9-32.

[19] Perry, M., & Herring, J. (2012). *GeoSPARQL - A Geographic Query Language for RDF Data*. Open Geospatial Consortium (OGC) Implementation Standard.

[20] Cox, S., & Little, C. (2017). *Time Ontology in OWL*. W3C Recommendation. https://www.w3.org/TR/owl-time/

[21] Guha, R. V., Brickley, D., & Macbeth, S. (2016). Schema.org: evolution of structured data on the web. *Communications of the ACM*, 59(2), 44-51.

[22] Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A Distributed Messaging System for Log Processing. *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB)*.

[23] KServe Community. (n.d.). *KServe: Highly scalable and standards based Model Inference Platform on Kubernetes*. https://kserve.github.io/website/

[24] Paszke, A., Gross, S., Massa, F., Lerer, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS) 32*.

[25] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. *ACM Queue*, 14(1), 70-93.

[26] Istio Authors. (n.d.). *Istio Service Mesh - Mutual TLS Authentication*. Istio Official Documentation. https://istio.io/latest/docs/concepts/security/#mutual-tls-authentication

[27] Amazon Web Services. (n.d.). *Amazon Simple Storage Service (S3) Documentation*. https://docs.aws.amazon.com/s3/

[28] Jonas, E., Schleier-Smith, J., Sreekanti, V., et al. (2019). Cloud Programming Simplified: A Berkeley View on Serverless Computing. *arXiv preprint arXiv:1902.03383*.

[29] IBM Cloud. (n.d.). *IBM Cloud Object Storage & SkyStore Architecture Documentation*. https://cloud.ibm.com/docs/cloud-object-storage

[30] European Space Agency. (n.d.). *Copernicus Emergency Management Service (EMS)*. https://emergency.copernicus.eu/