NEARDATA

Funded by the European Union

HORIZON EUROPE FRAMEWORK PROGRAMME

# NEARDATA

(grant agreement No 101092644)

## Extreme Near-Data Processing Platform

## D5.2 Final Validation in International Health Data Spaces

Due date of deliverable: 31-10-2025
Actual submission date: 31-10-2025

Start date of project: 01-01-2023 Duration: 36 months

# Summary of the document

| | |
|---|---|
| **Document Type** | Report |
| **Dissemination level** | Public |
| **State** | v1.0 |
| **Number of pages** | 99 |
| **WP/Task related to this document** | WP5 / T5.1, T5.2,T5.3,T5.4,T5.5,T5.6 |
| **WP/Task responsible** | BSC-CNS |
| **Leader** | Aaron Call (BSC) |
| **Technical Manager** | Daniel Barcelona (URV) |
| **Quality Manager** | Raúl García (DELL) |
| **Author(s)** | Aaron Call (BSC), Lorena Alonso (BSC), Andrés Benavides (BSC), Nicolas Gaitán (BSC), Pietro Morichetti (BSC), Daniel Barcelona (URV), Usama Benabdelkrim (URV), Raúl Gracia (DELL), Sean Ahearne (DELL), Alan cueva (DELL), Hossam Elghamry (DELL), Ger Hallissey (DELL), Sebastian Bodenstedt (NCT), Max Kirchner (NCT), Paolo Ribeca (UKHS), Andre Miguel (SCO), Maciej Malawski (SANO), Piotr Kica (SANO), Jan Przybyszewski (SANO), Sabina Lichołai (SANO), Kamil Burkiewicz (SANO), Jan Meizner (SANO) |
| **Partner(s) Contributing** | BSC, URV, DELL, UKHS, NCT, SANO |
| **Document ID** | NEARDATA_D5.2_Public.pdf |
| **Abstract** | This report will describe the final results from validations of the NEARDATA platform in the five use cases. We will publish data connector libraries, open datasets and sample workflow experiments. User experiences from the different DataSpaces. |
| **Keywords** | Use cases, Data connectors, Data throughput, Genomics, Epistasis, Genome-Wide Discovery, Metabolomics, Transcriptomics, Lithops, Pravega, SCONE, Lithops-HPC, Dataplug, HPC, Supercomputing |

# History of changes

| Version | Date | Author | Summary of changes |
|---|---|---|---|
| 0.1 | 05-06-2025 | Aaron Call | Table of contents. |
| 0.2 | 30-09-2025 | All | First draft. |
| 1.0 | 31-10-2025 | Aaron Call | Final version. |

## Table of Contents

## List of Abbreviations and Acronyms

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **API** | Application Programming Interface |
| **AWS** | Amazon Web Services |
| **CAS** | Configuration and Athestation Service |
| **CC** | Creative Commons |
| **CSV** | Comma-separated values |
| **DAG** | Direct Acyclic Graph |
| **DoA** | Description of the Action |
| **DOI** | Digital Object Identifier |
| **DoW** | Document of Work |
| **DTW** | Dynamic Time Wrapping |
| **ETL** | Extract-Transform-Load |
| **FaaS** | Function-as-a-Service |
| **FL** | Federated Learning |
| **GWD** | Genomic-Wide Discovery |
| **HPC** | High-Performance Computing |
| **KPI** | Key Performance Indicator |
| **MDR** | Multi Dimensionality Reduction |
| **ML** | Machine Learning |
| **MN5** | MareNostrum 5 |
| **T2D** | Type 2 Diabetes |
| **TEE** | Truested Execution Environment |
| **VCF** | Variant Call Format |

# 1 Executive summary

Deliverable 5.2 comprises the final release of KPI benchmarks on all use cases and data connector libraries. Essentially, this deliverable explains the outcomes of the project via its use cases and results achieved.

In this deliverable, we present the different challenges we faced during the project and how we addressed them through the implementation of data connectors. Such connectors are software units designed to tackle such issues in a fashion that is reproducible and reusable for similar use cases in the future.

We must note that the use cases are very different among them: this was intended to provide a wide representation of the life sciences ecosystem. In this manner, the connectors can be used with minimal effort by other use cases in their contexts. However, for this precise reason, the connectors are not used on other use cases (except in very specific situations), as that would not make sense.

In this deliverable, we explain the use cases we have selected to demonstrate the achievements of the NEARDATA's goals as well as the data connectors developed to achieve such goals. Those achievements are evaluated in terms of the following Key Performance Indicators (KPIs) as defined in the Description of the Action (DoA):

- **KPI-1** - Significant performance improvements (data throughput, data transfer reduction) in Extract-Transform-Load (ETL) phases validated with near-data connectors over extreme data volumes (genomics, metabolomics).
- **KPI-2** - Significant data speed improvements (throughput, latency) in real-time video analytics validated using stream data connectors.
- **KPI-3** - Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows.
- **KPI-4** - High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.
- **KPI-5** - Demonstrated simplicity and productivity of the software platform validated with real user communities in International Health Data Spaces.

Table 1 summarizes the major KPIs achieved by each of the use cases.

| KPI | Use Case | Summary of results |
|---|---|---|
| KPI-1 | Epistasis | Lithops-HPC connector improves data ingestion in GWD pipeline by **36x**. |
| KPI-2 | Surgery | Semantic video search latency is around **30**ms for large collections of surgical video. |
| KPI-1 | Transcriptomics | Early stopping technique have increased alignment throughput by **19.5%**. |
| KPI-1 | Genomics | FaaStream is up to **65.14%** cheaper than Flink running the genomics `Kpop` job. |
| KPI-1 | Metabolomics | Depending on the size, we get a speed-up on processing time ranging from **1.13x to 1.22x** faster. |

Table 1: Summary of most significant KPIs achieved on the use cases

Finally, we describe the integrations done with the different core architectures of the project described in Deliverable 2.3. Architectures such as Lithops, SCONE, or Pravega have made the use cases more efficient in terms of data throughput and/or transfer, as well as reduced latency or improved user experience. Additionally, as a result of this work, a new architecture was devised:

Lithops-HPC, which is explained in Deliverables 2.3 and 3.2. Such architecture has not only improved the performance of the genomics epistasis use case but also significantly improved the user experience of non-technical users, who do not need to provide specific details when interacting with the HPC supercomputer management system.

In summary, for each of the use cases we describe:

- A brief description of the use case.
- The Data Spaces used and/or provided.
- The targeted KPIs.
- The challenges addressed.
- The data connectors developed to solve the challenges.
- How AI has helped improve the use case, which context is Task 5.1 of the DoA.
- The integration of the use case with NEARDATA architecture components.
- The results and KPIs achieved.
- The collection of repositories made available (use case, connectors or utilities).
- The conclusions the project allowed to extract for the use case.
- The future work to be carried out after the project concludes.

This document presents an overview of the evaluation conclusions of each of the benchmarks, as well as a list of data connectors. We also provide a set of GitHub URLs where the use cases' source codes are made available, as well as the necessary architecture frameworks.

# 2 Use Case: Genomics Epistasis

## 2.1 Short Description

This use case[1] addresses the development of a high-performance computational pipeline aimed at uncovering groups of genomic variants associated with complex disorders at a genome-wide level (GW). Identifying multiple variants linked to such conditions is essential for deepening our understanding of underlying molecular mechanisms. However, the exhaustive analysis of all possible variant combinations remains a significant challenge, primarily due to the immense volume of data that must be ingested, stored, processed, and analysed concurrently. Overcoming this bottleneck requires the integration of scalable, dynamic, and distributed computational solutions capable of efficiently managing complex analytical workloads. The proposed workflow initiates with the ingestion and partitioning of large-scale genomic datasets, enabling the orchestrated combinatorial analysis through machine learning techniques.

## 2.2 Data Spaces

To address this challenge, we employed two large-scale genomic datasets for the analysis of Type 2 Diabetes (T2D):

- **70KforT2D**: 1.29 TB; $1.5 * 10^6$ million variants; 22,802 paired case-control individuals
- **UK Biobank**: 3.29 TB; $1.5 * 10^6$ million variants; 55,438 paired case-control individuals

These datasets, along with other genomic resources, are available through established public repositories, biobanks, and data spaces such as the Database of Genotypes and Phenotypes (dbGaP), the European Genome-phenome Archive (EGA), and the UK Biobank. Particularly, the individuals included in the 70KforT2D belong to five studies: Resource for Genetic Epidemiology Research on Aging (GERA), Finland-United States Investigation of NIDDM Genetics (FUSION), Wellcome Trust Case Control Consortium (WTCCC), Gene Environment Association Studies initiative (GENEVA), Northwestern University NUgene project (NUgene). The genetic information is publicly available through the dbGaP platform for FUSION (phs000867.v1.p1), GENEVA (phs000091.v2.p1), NUgene (phs000237.v1.p1), GERA (phs000788.v2.p3), and the Sanger platform for WTCCC. The UK Biobank information is available through the UK Biobank portal (https://www.ukbiobank.ac.uk/). Due to the sensitive nature of genomic data, access to genotype information typically requires a formal data access agreement between the data custodians and the requesting researchers, ensuring compliance with ethical and legal standards.

## 2.3 Targeted KPIs

The KPIs targeted in this use case correspond to:

- **KPI-1**: there is a need to improve the data ingestion throughput, to enable the reading of the entire genomics datasets within a reasonable amount of time.
- **KPI-3**: since there is a lot of data and multiple variants that come to it, converting this use-case into an extreme-data problem, as already discussed in deliverable D5.1, we require setting up an architecture that allows for scaling to an extreme, as well as considering the trade-off between resources required and available.
- **KPI-5**: the use case is typically managed by non-technical users (i.e, not coming from the computer science world) who are not experts on HPC resource management. However, supercomputers typically expect a certain level of knowledge, which turns out not to be the case, making them hard to manage. Therefore, another goal of the use case is to develop the solutions in a user-friendly fashion for the end user, while ensuring proper scaling and throughput.

[1] This use case's scientific side is described throuhgouly in the pre-print paper "Genetic Profiling and Early Detection of Type 2 Diabetes Subtypes through Sex-Stratified GWAS and Explainable AI" by Alonso et al. available at `https://www.medrxiv.org/content/10.1101/2025.07.24.25332120v1.full`. The work in this section comprises Tasks 5.1 and 5.3.

### 2.4 Challenges addressed

The implementation of this high-performance computing (HPC) genomic pipeline presents challenges both at the methodological and technical levels. From a research perspective, it requires the development of a tailored approach that leverages a combination of machine learning (ML) models to identify groups of genomic variants contributing to disease onset and progression. Technically, the pipeline demands the integration of specialized tools capable of supporting the full data lifecycle, including ingestion, storage, processing, orchestration, and analysis, to efficiently manage and interpret large-scale genomic datasets.

- **Machine Learning**: Traditional association testing methods often overlook the combined contribution of multiple genomic variants to disease development. Machine learning (ML) approaches have emerged as promising tools for identifying variant groups and assessing their impact on disease risk. However, the structure of genomic datasets, characterized by millions of features across a relatively limited number of individuals, poses significant challenges for the effective application of ML techniques. To address this, we implement a combinatorial strategy that aligns with the statistical constraints of ML methodologies. The genomic data is first partitioned into manageable chunks that conform to the computational limitations of ML tools, ensuring robust analytical performance. These chunks are then paired and analysed for their association with the disease. This approach enables scalable, genome-wide analysis and contributes to enhanced understanding of disease mechanisms, with potential implications for early detection and precision medicine.

- **Ingestion and processing**: The substantial volume of data stored in genomic datasets, often encompassing tens of millions of variants across hundreds of thousands of individuals, presents a significant challenge for efficient data ingestion. To ensure optimal performance and avoid computational bottlenecks, it is critical to minimize redundant read operations from the same file. This is particularly relevant when working with formats such as Variant Call Format (VCF), Binary Call Format (BCF), or scalable alternatives like Apache Parquet, which support columnar storage and compression. To address these challenges, the ingestion process is designed to partition the data into discrete chunks that can be processed in parallel. This approach reduces I/O overhead and enables distributed execution using Lithops in combination with DataPlug, facilitating efficient data access and transfer and, thus, allowing the scalable ingestion and processing of large genomic datasets.

  On the other hand, genomic data is sequentially structured and often distributed across multiple large files. To ensure generalization and model performance, it is necessary that the chunking strategy incorporates data shuffling, and to ensure that the shuffle of the data across files is efficient. Traditional shuffling methods require loading all files into memory, performing the shuffle, and writing the results back to storage, an approach that is both time and resource-intensive. The challenge is further compounded when experimenting with different chunk sizes to optimize model training, as each configuration demands additional processing and storage overhead. To tackle this problem, a data shuffle connector has been integrated into DataPlug to make this process transparent and integrated into the reading of data.

- **Orchestration**: Once the use-case becomes deployed in a parallel fashion and leveraging Serverless paradigms into HPC infrastructures, resource allocation becomes challenging. In such environments, resources are highly demanded, and so the more requested the longer the wait for them to be allocated, impacting the time to service. Thus, one must consider whether to request the optimal amount or request less in order to improve overall time to service. To mitigate this issue, we have developed an auto-scaling mechanism for Lithops-HPC that handles dynamic allocation of resources based on a forecast of available resources in the underlying system (i.e, supercomputer and/or HPC infrastructure). This is deeply described in the following section 2.7.

- **Analysis**: Conventional machine learning workflows typically begin with a hyperparameter tuning phase, followed by model training and evaluation. While this sequence is essential for optimizing predictive performance, it becomes computationally intensive when scaled to millions of executions, even within a parallelized and scalable infrastructure. To mitigate this challenge, we have developed a methodology (detailed in Section 2.7) that enables the identification of optimal hyperparameters across all datasets in advance. This innovation allows us to bypass the tuning step entirely, resulting in a substantial reduction in computational overhead. Furthermore, our combinatorial strategy yields multiple feature subsets and a diverse array of predictive models. To transform these outputs into a consolidated set of disease-associated variants and a unified predictive framework, we have implemented bespoke analytical techniques, also described in Section 2.7.

### 2.5 Data connectors developed

- **Data shuffle and partitioning connector**: this connector reads the input data, shuffles its contents, mixing the variants, and splits into several partitions. This provides partitions that are already disordered and can be later used by the data analytics engine that has been implemented.

- **Auto-scaling connector**: this component leverages a time-series-based forecasting model, predicting resources available in supercomputers, to later auto-scale the amount of resources allocated to the Lithops-HPC backends. This achieves dynamic resource handling of the infrastructure transparently and efficiently. With this connector, the users no longer need to decide resources for themselves.

- **ML accelerator connector**: this component allows running the required random forest analytics based on XGBoost on multiple GPUs, splitting the data. For this, Dask library has been merged together with the Data Plug partitioner.

- **HPC connector**: this connector is the main component of Lithops-HPC. It connects the Lithops architecture with HPC platforms, enabling not only the usage of HPC computing resources but also handling heterogeneous supercomputing systems. This allows us to dynamically run on GPUs and CPU partitions, as well as it would enables running on other devices (i.e., FPGAs) when available.

#### 2.5.1 Data shuffle connector

To overcome the machine learning challenges related to data shuffling and partitioning, we have developed an innovative data connector designed to facilitate the ingestion of genomics data for AI model training. This connector enables the reading of one or multiple text-based files (including VCF format), automatically shuffling their contents during read operations and generating arbitrary-sized chunks dynamically, without the need for intermediate storage files. The connector is implemented as a new format extension within the Dataplug[2] library. It encapsulates one or more VCF files and leverages Dataplug's capabilities to read data in slices of arbitrary size at runtime. This functionality is made possible through Dataplug's concept of meta-slices (refer to the Dataplug library documentation in D2.3 and D2.2). The format incorporates a data shuffling mechanism that ensures slice access lines in a randomized order while maintaining complete coverage of the dataset across chunks. During Dataplug's preprocessing phase, the connector indexes the number of lines and their corresponding byte positions. This index is stored in the object's metadata and enables efficient random access to lines using byte-range read operations, thereby eliminating the need for intermediate file creation.

Performance evaluations indicate that slicing with this connector is as fast or faster than traditional partitioning methods that do not involve shuffling. The improved performance is attributed

[2]DataPlug is further explained in Deliverable 2.3

to the index, which provides precise byte offsets for line starts, allowing direct access without scanning for line endings. However, this efficiency comes at the cost of increased preprocessing time and metadata size. For example, indexing a 4 GB dataset requires approximately two seconds, and a naïve indexing approach that stores a pointer for each line results in metadata of roughly 700 KB for a 4.5 GB dataset.

In summary, our data connector offers a scalable and efficient solution for shuffling and chunking genomics data during ingestion, thereby enhancing the training of AI models while minimizing resource consumption and storage overhead.

#### 2.5.2 Auto-scaling connector

This component leverages a time-series-based forecasting model, predicting resources available in supercomputers, to later auto-scale the amount of resources allocated to the Lithops-HPC backends. This achieves dynamic resource handling of the infrastructure transparently and efficiently. With this connector, the users no longer need to decide resources for themselves. This connector is further explained in section 2.7.

#### 2.5.3 HPC connectors

This is composed mainly of two connectors that are further described in Deliverable 3.2. The main components are:

- **ML accelerator connector**: This component allows running the required random forest analytics based on XGBoost on multiple GPUs, splitting the data. For this, the Dask library has been merged together with the Data Plug partitioner.
- **HPC connector**: This connector is the main component of Lithops-HPC. It connects the Lithops architecture with HPC platforms, enabling not only the usage of HPC computing resources but also handling heterogeneous supercomputing systems. This allows us to dynamically run on GPUs and CPU partitions, as well as it would enables running on other devices (i.e., FPGAs) when available.

### 2.6 AI-Enabled Variant Discovery: Combinatorial Machine Learning Approach

In this use case, Artificial Intelligence (AI) is applied to address the complexity of identifying groups of genomic variants associated with disease risk. Unlike traditional statistical methods, which typically assess variants in isolation, machine learning (ML) approaches enable the exploration of combined effects across variant groups. However, the scale and dimensionality of genomic datasets, often comprising millions of features across hundreds of thousands of individuals, pose significant statistical and computational challenges for ML-based analysis.

To overcome these limitations, we implement a combinatorial strategy in which the dataset is partitioned into manageable chunks. These chunks are then pairwise combined to ensure that all variants are tested in interaction with one another. This approach allows for scalable analysis while preserving the capacity to detect complex variant associations.

Despite the scalability introduced by this strategy and the NearData project connectors, which facilitate parallelization and resource optimization, the problem remains computationally intensive. To address this, we introduce a complementary analytical solution:

1. **Hyperparameter Optimization under Combinatorial Constraints:** Hyperparameter tuning, typically performed via k-fold cross-validation, becomes prohibitively expensive when applied across combinatorial datasets. To mitigate this, we conducted randomized tests based on known genomic associations and evaluated model reliability across subsets. This enabled the clustering of hyperparameters according to performance metrics, allowing us to select optimal configurations without exhaustive search.

Beyond the core scope of the NearData project, additional research-driven solutions have been explored to address the problem through combinatorial approaches. As these developments are not

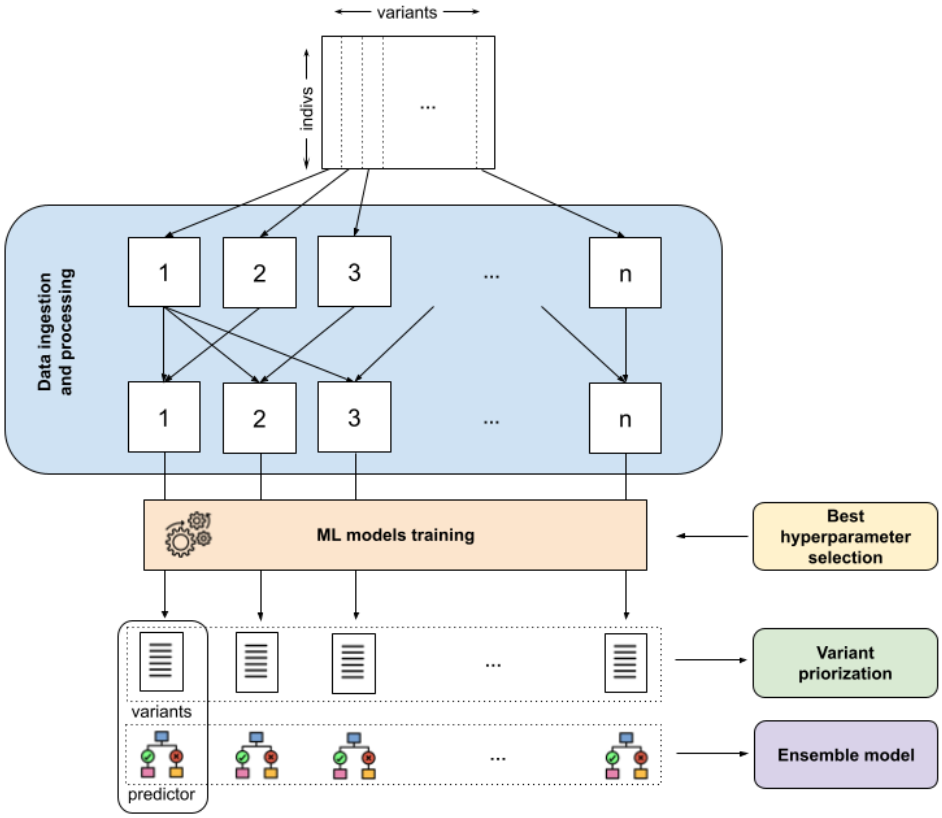



Figure 1: Genome-Wide Discovery overview

directly aligned with the project's primary objectives and do not exclusively impact the defined KPIs, we provide here a brief summary for completeness:

1. **Scoring and Prioritization of Variant Groups:** The combinatorial approach yields extensive lists of variant groups, many of which may be redundant or weakly associated with the disease. To prioritize meaningful results, we developed a scoring mechanism based on input-output frequency across executions. This preliminary method ranks variant groups by their recurrence and strength of association. Future iterations will incorporate graph-based models and stochastic processes to enhance prioritization and reduce redundancy.

2. **Ensemble Modeling for Predictive Robustness:** The combinatorial framework produces multiple predictive models rather than a single unified predictor. To consolidate these outputs, we implemented an ensemble learning strategy. Each model is treated as an expert predictor for a specific genomic region, and its reliability is assessed to assign a weighted contribution to the final ensemble. This approach improves overall predictive accuracy and robustness in disease detection.

### 2.7 AI-driven Resources Autoscaling Connector

The Function-as-a-Service (FaaS) paradigm divides workloads into independent tasks, each one processed by a dedicated CPU worker. When the number of tasks surpasses the system's available computational resources, an overload condition occurs (aka: demanding CPU Workload), leading to a task-to-worker imbalance. In such cases, pending tasks are placed in a workload queue, which is continuously monitored and iterated as computational resources become available, ensuring efficient task execution. In other words, demanding only available resources ensures that HPC job schedulers

(namely Slurm in our case) will grant resources much quicker than they could possibly do if we requested more than is actually available. The latter is a typical occurrence when users run their jobs requesting resources they really need without taking into account availability.

Users don't ask for more than is available for no reason. They do so because they typically employ MPI paradigm, which operates with a static set of resources. Consequently, if they did ask less, although the job would start sooner, it would still be completed much later than if it had waited in the first place.

To overcome these issues, Lithops-HPC is further extended with an autoscaling mechanism to dynamically adjust resources in the backend.

We propose to manage the resources of Lithops-HPC with the knowledge of the overall resources' availability in the system. This mechanism allows us to scale down resources as well, so we can have as many resources in the system as are available, as well as to scale down when those are not predicted to be required anymore. This autoscaling mechanism is what we call FaaSTs and explain in this section.

With FaaSTs we can forecast how many resources will be available in the next period of time - parametrizable - and acquire them faster as we request resources that are already available. To do this forecast, we employ a time-series model that gathers publicly available data of the system and allows us to do the prediction.

Lithops-HPC enables users to structure their applications as a collection of Python functions, which are efficiently distributed across multiple CPU workers. With the time series model integrated, it provides the *scaleUP* and *scaleDown* options, allowing to append (or release) workers into a deployed compute backend. FaaSTs uses these options to effectively modify the CPU resources in real time while also considering the HPC variations reported from the Time Series model, according to a *Scaling Policy*.

This is a novel policy to decide if increase or decrease the number of workers for a given computing backend. The Scaling Policy takes into account the HPC system status by setting three levels of system utilization, in terms of resources used:

- High - when the HPC system usage is over 80%[3]
- Moderate - when the HPC system usage is between 40% and 80%;
- Idle - when the HPC system usage is under 40%[4].

According to the status of the HPC system, the number of workers can increase or decrease by a certain percentage. In the situation of the HPC system in state "High", the number of workers is reduced to 10%, while if the state is "Moderate" or "Idle" the number of workers is increased of 10% and 20% accordingly [5]; this variation percentage is here called *Scaling Factor*. The Scaling Factor is then used to compute the actual number of workers according to the equation 1.

$$W_{i+1} = \min\left(\max W, \max\left(\min W, W_i \cdot (1 + \mathrm{SF}_i)\right)\right) \quad (1)$$

Where maxW and minW are arbitrarily chosen as upper and lower bounds for the size of the Lithops backend, and, in the case of this paper, it was decided for $maxW = 10000$ and $minW = 1000$ just for experimental purposes.

In conclusion, based on the HPC system status, the Scaling Policy applies a scaleUp or scaleDown with a certain Scaling Factor.

### 2.7.1 System architecture

Figure 2 presents the Function as a Service and Time Series (FaaST) architecture.

FaaSTs employs Lithops as *Compute Backend* (also called Lithops backend)(grey block at the top) to execute the client functions, which can be dynamically resized at most convenience, as it will be explained later in the paper. The *Web Scraper* (green block) is a web application listening for HTTP

[2]This threshold was chosen based on empirical observations, and it can be adjusted accordingly to the use case.

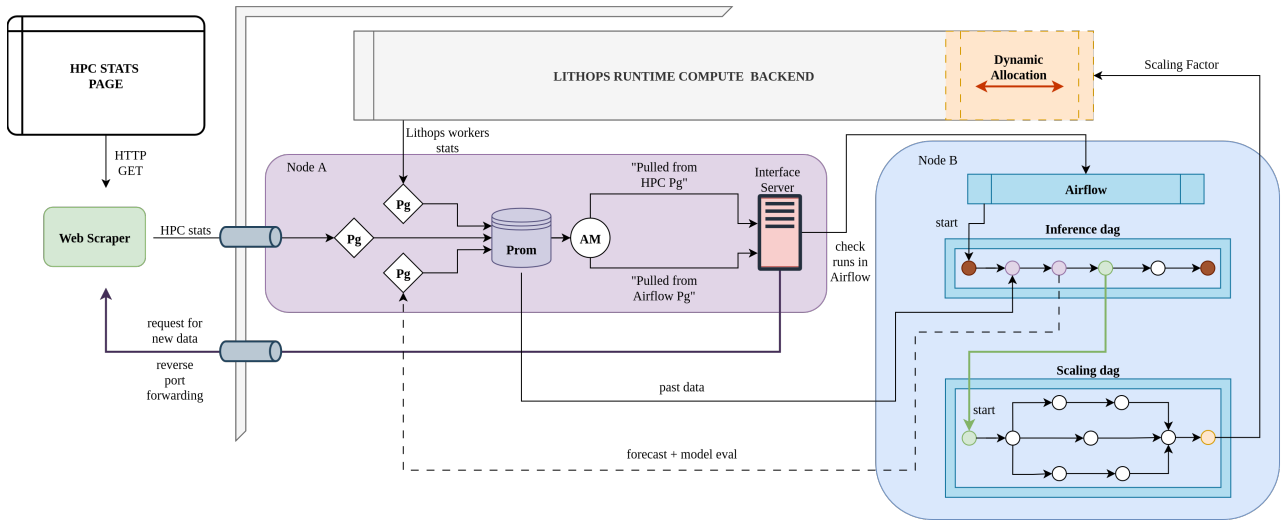



Figure 2: A simplified FasSTs architecture

requests and providing HPC system stats by scraping a dedicated webpage. Based on the way it is designed, a *Prometheus* database [1] (inside violet block) checks, at a given frequency (or scrape frequency), target objects in order to collect telemetry data that is being exposed. In this context, the target entities are three Prometheus Pushgateways ([2], [3], [4]), which serve as data containers populated with telemetry data from various architecture components. For instance, the Web Scraper component populates one Pushgateway with statistics from the HPC system (as illustrated in figure 2). The remaining two Pushgateways are used for inference and model evaluation tasks, and for collecting Lithops metadata originating from the Compute Backend, respectively. Another Prometheus component is the *Prometheus Alert Manager* [5] & [6] (again inside violet block), which catches the Prometheus alerts firing and automatically transmits a message to a certain destination: it works as a "notifier" informing that a certain event just took place. The Alert Manager has a single target to send the notification, this target is another web application, here called *Interface Server*. This application sends HTTP requests to the Web Scraper when Prometheus requires new data (according to the Prometheus scrape frequency). In addition, the Interface Server interacts with *Apache Airflow* [7], an open source framework to schedule and monitor workflows, and here it is used to run several pipelines like:

- Inference dag - this pipeline retrieves new data from Prometheus, transforms them if necessary, and provides them to the trained model for inference;
- Scaling dag - this pipeline uses the most recent prediction to assess the future state of the HPC system (meaning if the system is overloaded, stationary, or free) and dynamically allocate, reduce, or keep the current resources of the Lithops backend;
- Supplementary dags - this is a set of extra asset-aware pipelines designed to evaluate the goodness of the forecasting model in real-time and generate useful plots.

Specifically, the Scaling dag pipeline is triggered by the Inference dag if the last pipeline managed to produce and serve a valid prediction; otherwise, the Inference pipeline is considered as failed, and it is necessary to wait until the next scraping time, when Prometheus will request new data.

As the FaaST architecture is designed to leverage idle CPU resources, it implements the Scaling Policy: scaleUp and scaleDown to increase or decrease the number of Lithops workers in the Compute Backend. In the Scaling DAG pipeline, the *Scaling Factor* is computed based on the prediction of the HPC system status, which determines the percentage of Lithops workers to allocate or de-allocate from the Lithops backend.

With a clear understanding of the individual components of the FaaST architecture, Figure 3 offers a comprehensive depiction of the temporal sequence of interactions among the major components.

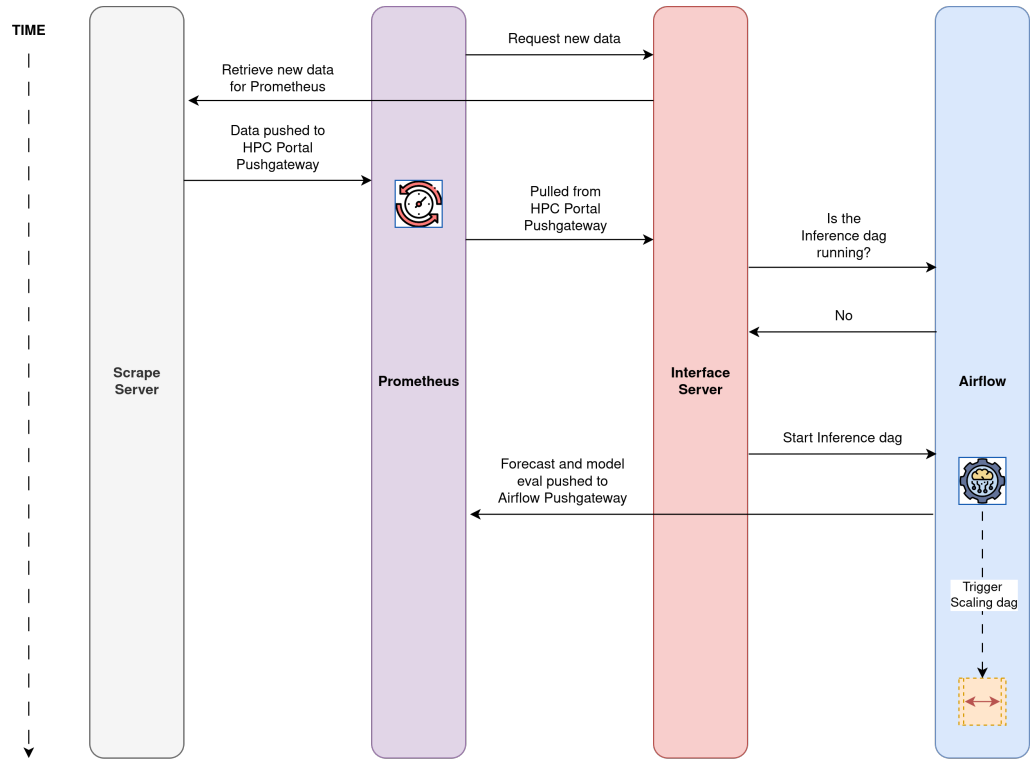



Figure 3: A simplified FasSTs workflow

Where, for simplicity, all the Prometheus components (meaning Prometheus DB, Prometheus Pushgateways, and Prometheus Alert Manager) are compressed inside Prometheus.

What is interesting to observe is that FaaST architecture is event-driven: by exploiting the Prometheus scraping interval (i.e., the frequency at which it pulls data from the target object), it is possible to trigger the forecasting model and, ultimately, the dynamic allocation of resources for the Lithops backend, without the need for an orchestrator.

### 2.7.2 Time series forecasting methodology

The preparation of the Statistical and Machine Learning model relied on the following phases:

1. Data collection - system-related data (i.e., total available CPUs and the number of assigned CPUs at a given timestamp) were collected at 2-hour intervals over a period of one month. A total of 12 data points per day and 336 data points within a month were collected. Due to the mainly constant nature of the total available CPUs it is more of interest to focus on the number of assigned CPUs.

2. Data Exploration - prepossessed data was examined to understand their representation, meaning, and seasonality. Several analyses were applied to detect trends, seasonal components, and residuals. No relevant data points were removed during this phase.

3. Data Preparation - collected data were examined to extract insights and identify unseen patterns. Outliers were treated to refine the dataset and shape an appropriate waveform for the forecasting model. In order to ensure data stationarity, it is convenient to apply a differentiation technique that should remove trend and smooth seasonality (if applied to certain lags).

4. Time Series Model Selection - Seasonal ARIMA model (or just SARIMA) was chosen as the model requirements were met.

5. Model Finetuning - a greedy search algorithm was employed to identify the optimal configuration of SARIMA parameters. The total of possible parameter combinations was examined in parallel. The best configuration was conserved for the future stages.
6. Model Evaluation - final model was selected based on performance scores derived from three assessment categories, based on qualitative assessment of the model residuals and quantitative assessment prediction by checking the error computed.

About the modeling part, a seasonal ARIMA model presents a set of hyperparameters:

- p - explain correlation with past values (it depends on significant lags in PACF plot);
- d - make series stationary (it depends on whether the time series is stationary or not);
- q - explain correlation with past errors (it depends on significant lags in ACF plot);
- P - seasonal AR order;
- D - seasonal differencing order;
- Q - seasonal MA order;
- s - seasonal period.

The subset (p, d, q) is a proper subset of the ARIMA model, while the subset (P, D, Q, s) makes ARIMA addressing the seasonal part of the time series.

The SARIMA hyperparameters can assume any integer value between 0 to infinity; therefore set of all possible combinations is also infinite. However, thanks to the data exploration and data preprocessing applied it is possible to restrict the possible set of combinations to SARIMA(p,0,q)(P,0,Q,13) since it was previously applied the differentiation and there was recognized a daily seasonality was recognized. For the remaining hyperparameters (p, q, P, Q) it is possible to apply a greedy search algorithm over all possible values between 0 to 7 (larger could lead to overfitting), resulting in a set of more than 4000 combinations of hyperparameters.

The various SARIMA models are evaluated according to three categories of evaluation metrics: analysis of the residuals, prediction errors, and distances between real and predicted time series.

The residuals analysis is a qualitative assessment of the model's goodness, and it is used to check the reliability of the model based on some statistical tests over the residuals (meaning the differences between the actual observed values in a time series and the values predicted by the SARIMA model). The residuals need to hold some properties to ensure the model forecast is trustworthy. In particular, the residuals have to be: homoscedastic, which means the variance of these residuals is consistent across all observations; normally distributed to ensure the model catches the underlying pattern(s); and finally, the residuals do not show any autocorrelation.

The prediction errors are a quantitative assessment of the model's goodness, and it is composed of: Log-Likelihood, which is a way to understand how much of the data is explained by the model; BIC, AIC, and HQIC are from Game Theory and are used to balance model fit with complexity; while MAE, MSE, RMSE, and MAPE are different ways to measure the error computed from the model between the forecast and the actual value. Finally, a set of distance metrics as Dynamic Time Wrapping (DTW), Manhattan, and Euclidean distances.

### 2.8 Integration with NEARDATA architecture components

- **MPI**: Parallelized HPC processing.
- **Lithops**: Scalable data analytics and AI/ML workloads.
- **Lithops-HPC**: this software integrates Lithops into the HPC supercomputing resources, leveraging the HPC connector [8]. The full architecture is further described in NEARDATA deliverable 3.2.

- **Dataplug**: Dynamic partitioning tool.
- **RabbitMQ**: Workload manager.
- **GekkoFS**: Lightweight, temporary distributed file system for high-throughput HPC [9].
- **NVIDIA**: High-performance computing, deep learning, and AI workloads.

### 2.9 Results and KPIs achieved

In this section, we split the results in two, on one hand the results obtained by the auto-scaling method. On the other hand, the results obtained using Lithops-HPC on our use-case.

#### 2.9.1 Infrastructure used

In both situations MareNostrum 5 (MN5) supercomputer has used. MN5 combines Lenovo ThinkSystem SD650 V3 and Eviden BullSequana XH3000 architectures, providing two partitions with different technical characteristics. The MN5-General-Purpose Partition comprises 6.408 nodes based on Intel Sapphire Rapids (4th Generation Intel Xeon Scalable Processors). This general partition was used for the experiments described in the next lines.

Moreover, we have also used the accelerated partition of MareNostrum 5 whose main characteristic is that each node contains 4 NVIDIA H100 GPUs, which are one of the most powerful GPUs currently available. These GPUs are not built using traditional PCIe ports but rather are built on top of specialized motherboards and embedded to them, so the communication bus between CPU and GPUs is as fast as a memory-CPU bus.

Additionally for some experiments we also used Nord4, which is a portion of the previous supercomputers generation: MareNostrum 4.

#### 2.9.2 Auto-scaling methodology evaluation

In the following lines we assess the auto-scaling methodology. For this portion we have used MDR use-case. As a reminder and already explained in previous deliverable D5.1 MDR is a statistical approach used to identify pairs of genetic variants that, synergistically, contribute to the development of these disorders. However, MDR demands evaluating all possible pairwise SNP-SNP interactions, which can be computationally intensive. Equation (2) calculates the total number of such pairwise combinations.

$$\text{Total pairwise SNP combinations} = \binom{N}{2} = \frac{N \cdot (N-1)}{2} \tag{2}$$

where:

- $N$ is the total number of SNPs (variants) in the dataset.
- $\binom{N}{2}$ denotes the binomial coefficient, representing the number of unique, unordered pairs of SNPs.
- $O(N^2)$ denotes the computational Order.

Table 2 shows the synthetic datasets designed to evaluate the proposed strategy. They were inspired by the Northwestern NuGENE project cohort1.

#### 2.9.3 System-Level Forecasting

The HPC system usage is represented by the number of CPUs the system has available and the number of CPUs assigned to the users to run their applications as depicted in figure 4.

Figure 4 shows a snapshot of data from the supercomputer MareNostrum5. Data represents the number of CPUs currently available in the supercomputer, from 0 to a maximum of around 700.000 CPUs available on the y-axis, while on the x-axis, a temporal window of 1 month of data with a frequency of 2 hours. The orange line shows the total CPUs, which is the number of CPUs available

| | SNPs | # patients | Total Combinations (millions) |
|---|---|---|---|
| **big (UK)** | 15586493 | 422000 | 1.2E+08 |
| **small (70K)** | 1883192 | 1128 | 1.8E+06 |
| **tiny** | 100000 | 1128 | 5.0E+03 |

Table 2: Experimental Datasets

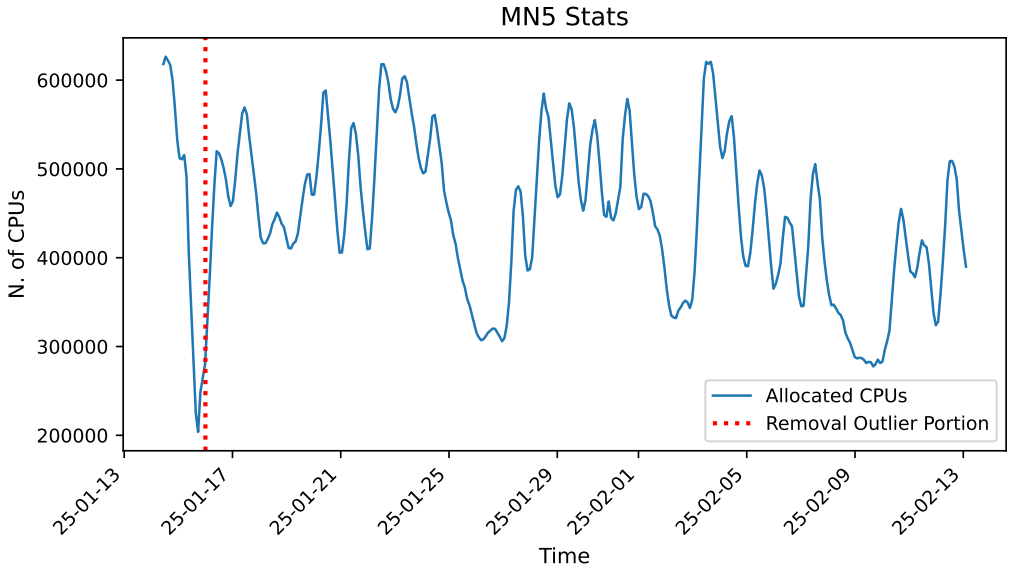



Figure 4: Available and allocated CPUs over time

and ready to be assigned to a user of the supercomputer, while the blue line shows the allocated CPUs, which is the number of CPUs already in use and allocated to users' jobs. Finally, the vertical red dashed line highlights a potential outlier in the time series, due to its extreme value.

The allocated CPUs time series can be further analysed by applying a Time Series Decomposition. Figure 5 shows the Time Series Decomposition into three subplots, which share the x-axis representing the number of points ordered in temporal order.

At the top is shown, in blue, the trend component of the allocated CPUs time series, where the y-axis is the number of CPUs in the supercomputer. The subplot in the middle shows, in yellow, the seasonal component of the allocated CPUs with a frequency of 13 data points, representing almost a daily seasonality with a variation between $\pm$20.000 CPUs as shown on the y-axis. Finally, the subplot at the bottom shows, in red, the residual component of the allocated CPUs time series, which is randomly fluctuating between $\pm$100.000 CPUs, as shown on the y-axis.

A different analysis to focus on the relation among the data points is provided by the autocorrelation and partial autocorrelation plots (figure 6). The figure is composed of three subplots, for each of which, the x-axis represents the lags from the most recent data point (lag 0 is related to the current data point at time t, lag 1 is the data point at time t-1, lag 2 is the data point at time t-2, and so on). While the y-axis shows the autocorrelation between the current data point at time t with a certain lag, and this value can vary in the range [-1, +1], where -1 means negative autocorrelation, 0 means no correlation, and +1 means positive autocorrelation.

In Figure 6, we present the autocorrelation and partial autocorrelation analysis of the allocated CPUs time series. The left subplot depicts the autocorrelation, where lag 1 exhibits a positive correlation of 0.25 and lag 13 exhibits a negative correlation of 0.35. The center subplot showcases the partial autocorrelation, where the current data point at time t is correlated with lags up to 40. Notably, lag

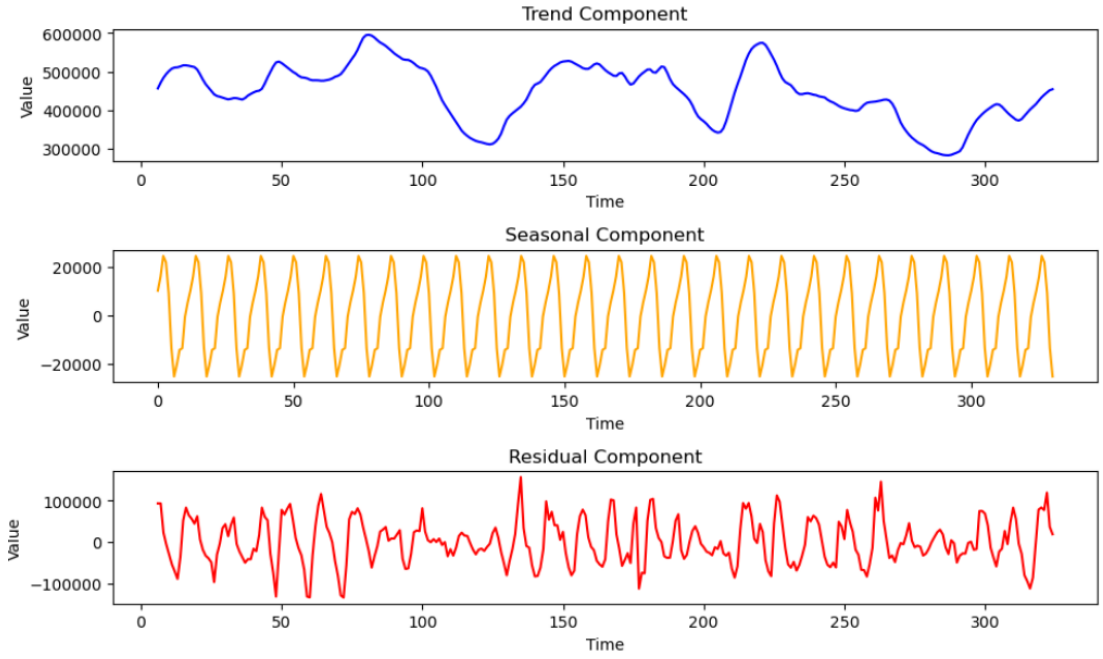



Figure 5: Allocated CPUs Time Series Decomposition

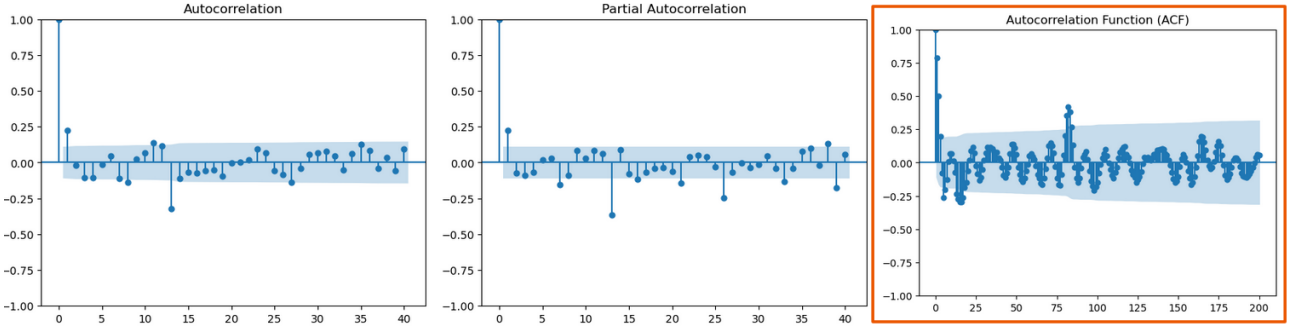



Figure 6: Autocorrelation and Partial Autocorrelation

1 exhibits a positive correlation, while lags 13, 26, and 39 exhibit negative correlations with varying magnitudes. The right subplot further extends the analysis up to lag 200, mirroring the information observed in the left subplot. However, additional lags are observed around lag 89 that exhibit positive correlations with the most recent data point.

As mentioned in 2.7.2, in order to ensure data are stationary, it is recommended to apply a differentiation.

Figure 7 includes two subplots: on the right is the differentiated allocated CPUs time series, where it was applied first order differentiation and differentiation at lag 13.

The second subplot on the right shows the Q-Q plot of the differentiated allocated CPUs time series, where on the x-axis as shown the theoretical quantiles in the range [-3,+3] and on the y-axis is shown the quantile values in the range [-200.000, +200.000]. The blue dots are the quantile distribution of the differentiated allocated CPUs compared to the red line, which represents the theoretical distribution of quantiles for the Normal distribution.

Several experiments were conducted to evaluate the performance of the SARIMAX model under various hyperparameter configurations. The results presented below correspond to the five best-performing configurations, i.e., the top-5 SARIMAX model settings.

Table 3 shows the Top-5 Seasonal ARIMA models among all the experiments conducted.

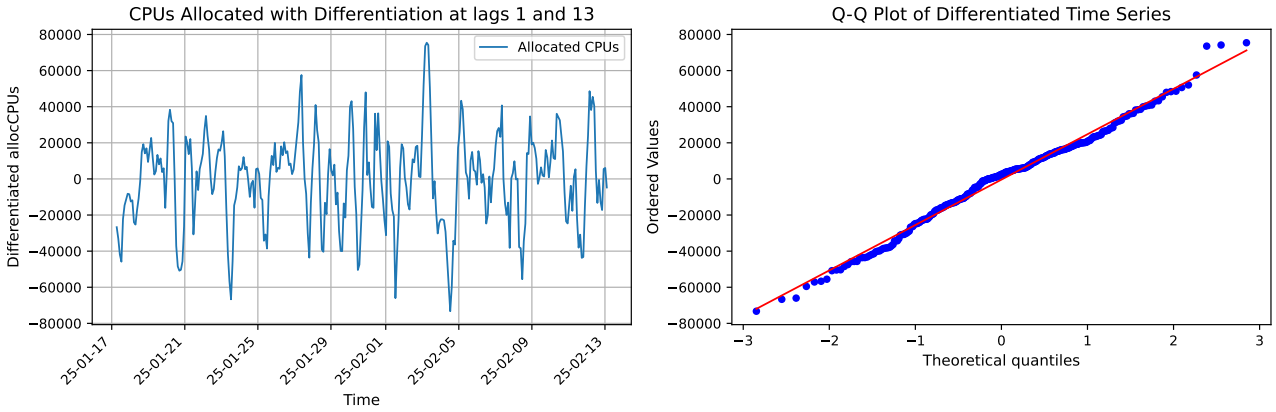



Figure 7: Time Series Differentiation and Q-Q plot

| ARIMA | | | Seasonal Comp. | | | |
|---|---|---|---|---|---|---|
| p | d | q | P | D | Q | s |
| 6 | 0 | 6 | 3 | 0 | 0 | 13 |
| 6 | 0 | 6 | 3 | 0 | 5 | 13 |
| 4 | 0 | 6 | 2 | 0 | 5 | 13 |
| 5 | 0 | 6 | 2 | 0 | 4 | 13 |

Table 3: Top-5 Seasonal SARIMAX hyper-parameters

Table 4 shows the Top-5 Seasonal ARIMAX models assessed on the set of prediction error metrics.

Finally, the distance between the actual and predicted time series is another set of metrics to quantitatively assess the goodness of the model. This category is composed of the following measures: Euclidean, Shaped-Based, Manhattan, Minkowski, Dynamic Time Wrapping (DTW) [10], Shaped DTW, Amerced DTW, Longest Common SubSequence (LCSS) [11], Edit Real [12], Move-Split-Merge (MSM) [13]. These metrics, differently from the prediction errors metrics, are more flexible since are less time-dependent (MSE and RMSE compute the error between the actual and predicted values according the same timestamp, it means that MSE and RMSE might show a large error, even if the predicted time series is just shifted forward of few data point, which is not a problem for DTW).

Table 5 shows the distance between the actual and predicted data points at the same timestamp. At the top of the table it is shown the Top-5 SARIMAX models by hyperparameters are shown, which are evaluated according to a set of distance metrics (in the left side of the table). Top-5 SARIMAX

| Prediction Error | | | | | |
|---|---|---|---|---|---|
| (p, q, d) (P, Q, D, s) | (6,0,6) (3,0,0,13) | (6,0,6) (3,0,5,13) | (4,0,6) (2,0,5,13) | (5,0,6) (2,0,5,13) | (5,0,6) (2,0,4,13) |
| Log-Like | -3425.04 | -3426.35 | -3441.72 | -3442.1 | -3442-04 |
| AIC | 6882.08 | 6886.7 | 6911.43 | 6914.19 | 6914.08 |
| BIC | 6904.4 | 6948.67 | 6962.47 | 6968.87 | 6968.76 |
| HQIC | 6905.46 | 6911.55 | 6931.89 | 6936.11 | 6936.0 |
| MAE | 31881.35 | 31626.39 | 34769.46 | 34987.73 | 35345.12 |
| RMSE | 37245.6 | 37098.07 | 40816.06 | 41100.51 | 41939.2 |
| MAPE | 260.61 | 242.99 | 286.94 | 279.24 | 265.32 |

Table 4: SARIMAX model Top-5 experiments based on Prediction Error category

models are sorted according to the minimum of the set of distance metrics, from left to right.

| Distance Error | | | | | |
|---|---|---|---|---|---|
| (p, q, d)<br>(P, Q, D, s) | (6,0,6)<br>(3,0,0,13) | (6,0,6)<br>(3,0,5,13) | (4,0,6)<br>(2,0,5,13) | (5,0,6)<br>(2,0,5,13) | (5,0,6)<br>(2,0,4,13) |
| Eucledian | 1.17 | 1.18 | 1.22 | 1.23 | 1.27 |
| Shape-Based | 0.59 | 0.61 | 0.72 | 0.72 | 0.71 |
| Manhattan | 6.88 | 6.95 | 7.19 | 7.33 | 7.56 |
| Minkowski | 1.17 | 1.18 | 1.22 | 1.23 | 1.27 |
| DTW | 0.96 | 0.97 | 1.21 | 1.16 | 1.11 |
| Shaped DTW | 1.37 | 1.34 | 1.48 | 1.52 | 1.42 |
| Amerced DTW | 1.37 | 1.39 | 1.48 | 1.52 | 1.62 |
| LCSS | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Edit Real | 0.61 | 0.61 | 0.67 | 0.66 | 0.59 |
| MSM | 6.88 | 6.95 | 7.19 | 7.33 | 7.56 |

Table 5: ARIMA model Top-5 experiments based on Distance Error

### 2.9.4 Auto-scaling with time series on Lithops-HPC

Table 6 shows the total number of combinations processed per worker per second for the different implementations using the MDR use-case.

| | Total combinations (millions) | | |
|---|---|---|---|
| **CPUs/Tool** | **Python Spark** | **MPI C** | **Lithops** |
| 112 | 4906 | 29582 | 10770 |
| 224 | 9812 | 59164 | 21540 |
| 336 | 14718 | 88745 | 32310 |
| 448 | 19624 | 118327 | 43081 |
| 560 | 24530 | 147909 | 53851 |
| 672 | 29436 | 177491 | 64621 |
| 784 | 34343 | 207073 | 75391 |
| 896 | 39249 | 236655 | 86162 |
| 1008 | 44155 | 266237 | 96932 |
| 1120 | 49061 | 295819 | 107702 |
| **COMBS/SEC/CORE** | **507** | **3057** | **1113** |

Table 6: Total combinations per second computed for a execution time of 24 hours (86400 seconds)

Figure 9 shows a hypothetical execution of MDR-Lithops execution with 200 workers, maintaining a minimum number of 100 active workers. It shows as the total of workers scale based on HPC's CPU fluctuations.

On the x-axis is represented the time, while on the 1st y-axis (on the left) it is shown the percentage of CPU occupation of the supercomputer, and on the 2nd y-axis (on the right) it is shown the number of CPUs included in the minimal set of active workers. The chart illustrates the temporal trend of the percentage of total CPU utilization of the supercomputer (blue line) alongside the trend in the number of CPUs exceeding the baseline threshold over time. The plot background shows additional information: as the percentage of CPU usage cross a 90% threshold, the HPC system is considered overloaded and therefore the total number of workers decrease as expected from the *Scaling Policy* introduced in section 2.7. This circumstance is well represented by the colored portions in the plot backend, where red is associated with the overloading status of the HPC system, yellow represents

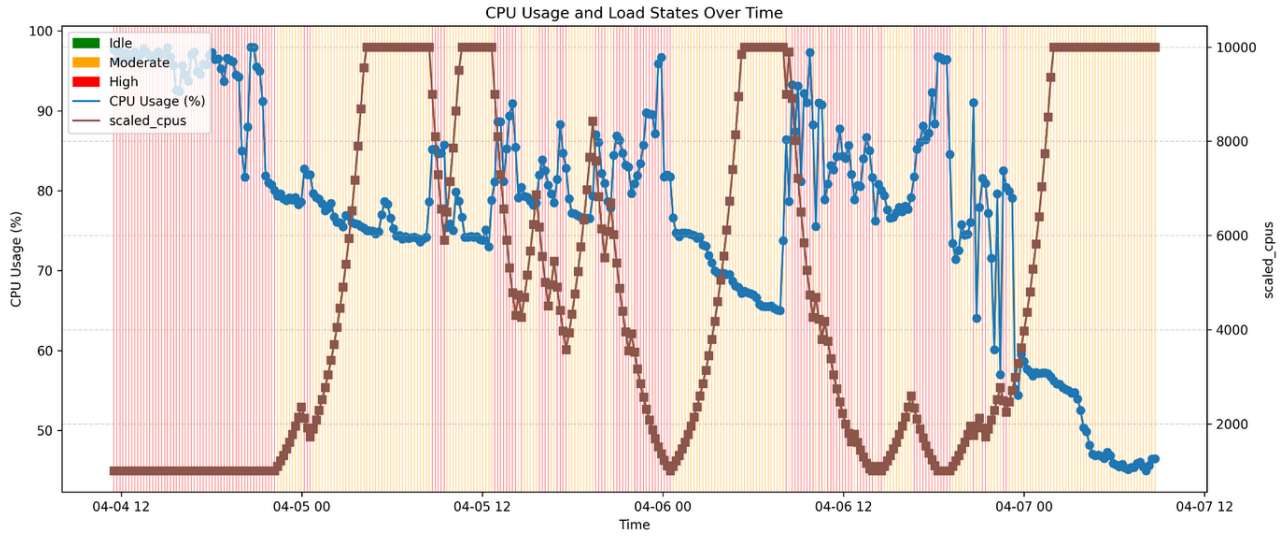



Figure 8: FaaST scaling execution using the MN5 cluster

a moderate level of overloading, and green for overloading at all. An additional aspect of figure 8 is given by the way the CPU Usage and number of workers lines are represented: here, the two lines assume a "square" line-style where each square is associated with a certain value (depending on the y-axis of reference). Focusing on the number of workers line, the distance (over the x-axis) between two consecutive squares should represent the scraping frequency of the Prometheus database and, consequently, the distance (in time) between one forecast and the successive forecasts from the Time Series predictive model. While each square does not only represent the total number of workers available (on top of the baseline), but also a specific SLURM job associated with the batch of new CPUs where to execute new Lithops workers. Therefore, each square in the brown line is actually a SLURM job running in the HPC system.

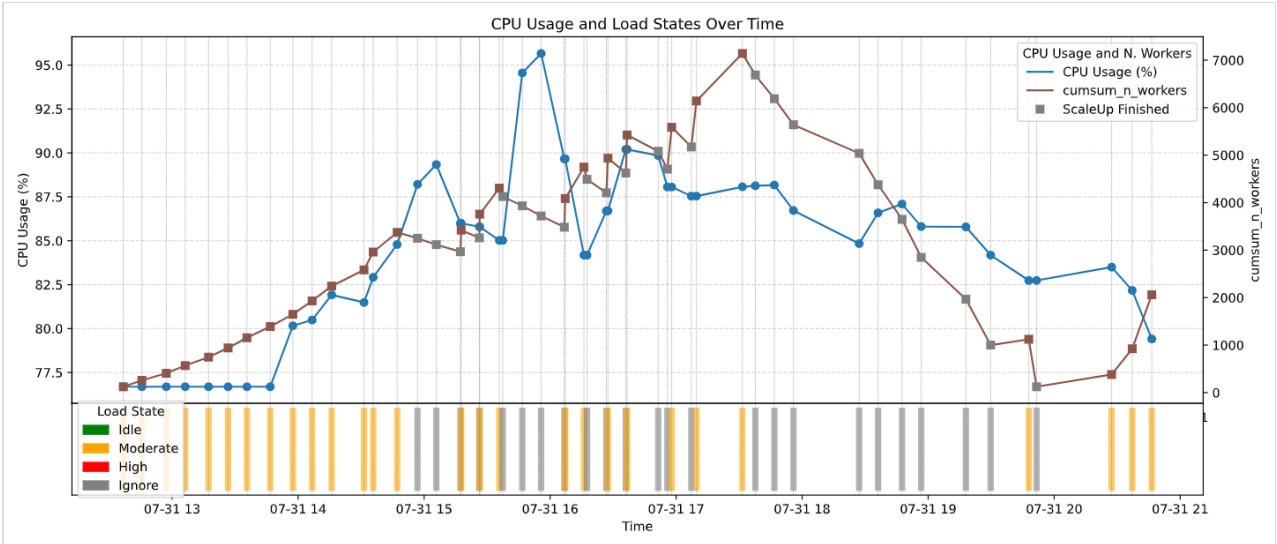



Figure 9: Experimental FaaST scaling execution using the MN5 cluster

A more realistic situation is given by figure 9 where the main differences with figure 8 are the following:

- clear representation of the *Scaling Down Policy* from the grey-squares on the brown line;

- not constant distance between consecutive brown-squares, possibly due to connection issue with the scrape webserver;
- status of the HPC system, previously represented with a background color, is now show as colored bars in a subplot.

Figure 10 shows the execution time of the same matrix multiplication problem for MPI, Lithops with a fixed number of available workers (hpc-baseline), and FaaST architecture with an extendable cpu backend (hpc-default). The amount of CPU's available for the three frameworks is the same and corresponds to 100 CPUs, except for FaaST, which could extend its CPU backend over time.

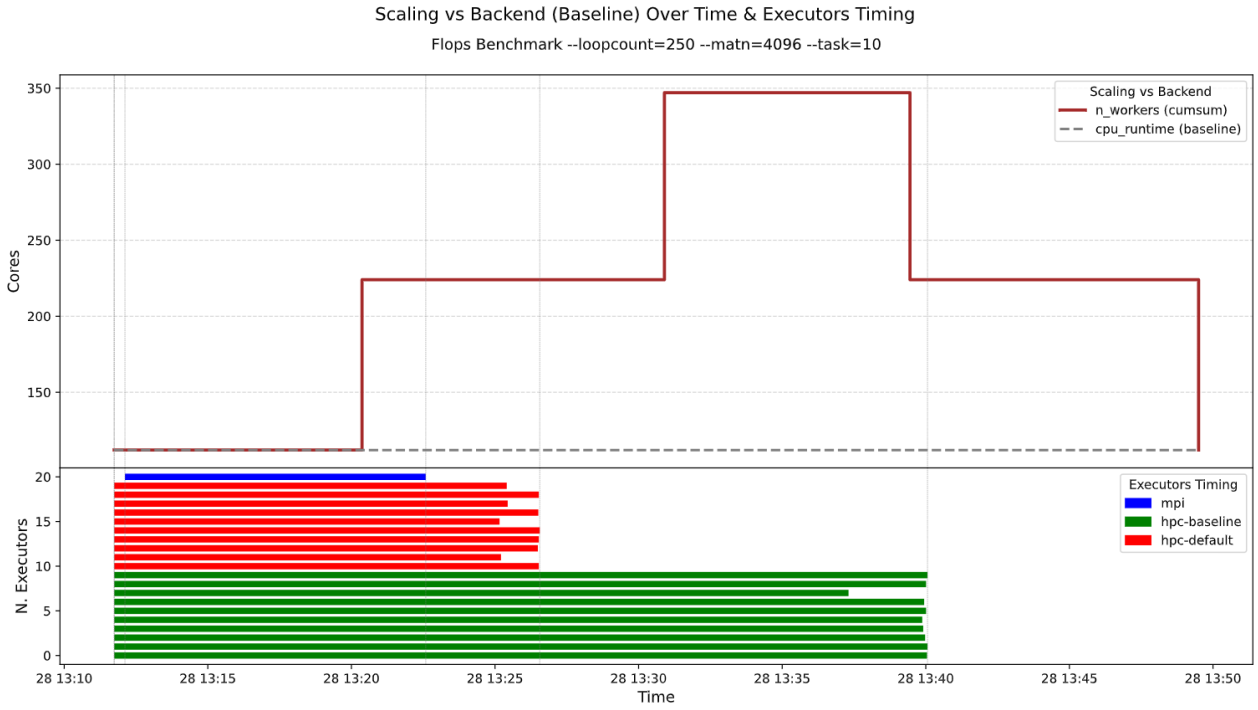



Figure 10: Execution time comparison among MPI, Lithops and FaaST Architecture

The execution time of the three frameworks is shown in the "Executors Timing" chart (at the bottom), where there is a single application using MPI (in blue), 10 applications are using the same hpc-baseline cpu backend (in green), and the other 10 applications are using the extensible hpc-default cpu backend (in red). On the y-axis is shown the ID of each application (regardless of what framework they are using), and on the x-axis the time, which is shared with the "Scaling vs Backend" chart (at the top). The top chart shows how the hpc-default backend is extended as soon as new workers are available (brown step-line) in comparison with the hpc-baseline dashed line, which remains fixed.

### 2.9.5 Auto-scaler experiments discussion

Given that each pairwise interaction is independent, they can be processed in parallel. It makes Function-as-a-Service (FaaS) paradigm ideal to decompose the workload into tasks and distribute them across multiple workers, enabling efficient batch processing. In this context, each task corresponds to a unique pairwise interaction, which is assigned to a CPU worker for evaluation. However, due to the vast number of interactions to be computed, the work queue may quickly exceed available resources (demanding CPU condition), leading to a long task queue and an unreasonable waiting time for MDR to finish.

For the forecast model to predict the HPC system overload, the analysis conducted reveals several insights from the data.

From figure 4 it is evident that the available CPUs is mostly constant and represents an upper bound for the allocated CPUs; therefore, it is more of interest to focus on the allocated CPUs time series. In fact, the allocated CPUs not only illustrate the number of CPUs assigned to users' jobs, but it seem to follow some kind of user pattern, for example, fewer CPUs are allocated outside working hours or during the weekend, where a decrease in magnitude occurs. In addition, the same plot highlights a rare situation around the 12th of January where both availCPUs and allocated CPUs decay immediately to zero, showing a strong fluctuation probably due to the outage of the HPC service. This was probably an exceptional situation that led to an anomaly in the supercomputer: similar circumstances may occur again in the future for unexpected disruption, maintenance, or outages and they should be taken into account while dealing with this specific type of data.

Another aspect of the data examined is whether data is stationary, i.e., whether the time series data shows any gradual upward or downward trends over time.

Stationarity is a property that is crucial for the application of certain statistical models, such as the Autoregressive Integrated Moving Average (ARIMA) model, which rely on the assumption of stationarity for accurate forecasting.

In other words, a stationary time series can be described as one that does not exhibit a gradual upward or downward movement over time. This implies that the mean, variance, and autocorrelation function (ACF) of the series remain constant over time.

The stationarity property is an essential prerequisite for utilizing models like ARIMA, which assume that the data exhibits no long-term patterns or trends. By ensuring stationarity, forecasters can increase the accuracy of their predictions and gain confidence in the reliability of their forecasts.

Further information can be derived from the Time Series Decomposition in figure 5: from the decomposition it is not clear if there is any trend involved, due to the no linearity of this component, different situation is for the seasonal component where it shows a strong and well-defined repetition representing the daily seasonality, as shown by the recurrent spikes. Finally, from the residual component, there should be no clear pattern, meaning that the decomposition successfully explains each component; however, from the plot, it is not clear if the residuals present some seasonality that is not fully captured by the seasonal component. For all three subplots in figure 5, the lag 0 is trivial since it is the most recent data point and it can be ignored, while all the lags falling inside the blue shady area are not relevant or barely relevant since the fall in the shady area (lags in this area are not statistically significant and can be ignored).

Further insights can be deduced from the data by looking at the autocorrelation and partial autocorrelation plot [14] of the allocated CPUs time series, where it is possible to infer the stationarity and seasonality of the data.

From figure 6, the autocorrelation plot shows a trend component, due to the magnitude of the first 10 lags, which tends to decrease lag by lag. This, in combination with the decomposition analysis, may suggest the presence of a weak trend in the data. For this reason, it is recommended to apply statistical tests to assess if the time series is trend dependent: the Augmented Dickey-Fuller test [15] report a value of -4.465770 (p-value = 0.000227), while the KPSS test [16] report a value of 0.602767 (p-value = 0.022385); for both cases the p-value does not cross the probability thresholds of 0.05 and therefore both test suggest the stationarity property of the time series. Nevertheless, considering the results from the statistical tests, the autocorrelation analysis and time series decomposition, it cannot be ruled out that the possibility of a weak trend needs to be taken into account.

Again, from 6 it is evident that the seasonal nature of the data is due to the sinusoidal shape of the lags. In fact, the partial autocorrelation plot reveals that:

- lag 1 is clearly significant, which means any new data point has a strong dependency on the most recent past data;
- lags around lag 12 are significantly relevant, which suggests daily seasonality, probably due to the HPC users' behavior, which tends to use the HPC system during working hours;

- considering a much larger window, it is clear that also lags around lag 89 are also significant, which reveal a weekly seasonality.

Data stationarity is an important property for any time series. For the allocated CPUs, it was achieved by applying a double differentiation to remove trend and daily seasonality, as shown in figure 7, while the weekly seasonality is converted to an exogenous variable, as Fourier terms, for the SARIMAX model.

The differentiated allocated CPUs is still a similar time window of 1 month on the x-axis, but this time the number of CPUs varies in the range [-150.000, +150.000]. As for the Q-Q plot, the differentiation had a positive impact, where, now, data are mostly normally distributed, but extreme quantiles tend to deviate from the normality assumption; therefore, it might be appropriate to check for outliers. There are many ways to perform outlier detection but they depend on the data shape and properties. In this case, the Modified Z-Score [17] is used since the normality assumption is not guaranteed and it is more robust against outliers. The Modified Z-score revealed 3 outliers: 2025-01-31, 2025-02-01 and 2025-02-07, which can be mitigated using the mean of the values around the outliers, in order to preserve the information on these points.

The preprocessed allocated CPUs time series is used to fit multiple ARIMA models, the top-5 shown in table 4 according to the prediction error and table 5 according to the distance error are evaluated by measuring the error computed between the forecast and the actual values. From the tables, it is clear that the ARIMA(6,0,6)(3,0,0,13) is the best in most of the evaluation metrics (both in prediction and distance tables). However, even if this is the best model, it still computes a relevant error as it is clear from the MAE and RMSE that the error computed is around 35000 CPUs from the real number of CPUs allocated in the near future. Not much can be said looking at 5, where the differences between the best ARIMA model and the rest seem minimal.

Concerning the experiments conducted in section 2.9.4, figure 8 shows an ideal behavior of the FaaST architecture; however, it does not take into account some events that may deflect the total number of workers line:

1. Newly available workers are associated with a Slurm job and each Slurm job has a planned completion time (meaning the job is supposed to complete after a certain amount of time). This will result in the Slurm job being de-allocated and, consequently, the workers will be lost.

2. FaaST architecture is empowered by Airflow and the allocation of a Slurm job is associated with a specific Airflow Task. If the Slurm job remains pending for too long, Airflow will cancel the Airflow Task, throwing an error.

3. A Slurm Job can be canceled due to the CPU percentage usage crossing the threshold, meaning that the workers associated with the Slurm job are forced to terminate.

4. Anything else that could prevent the FaaST Architecture from deploying new workers, for example, temporary disconnection with the scrape webserver running outside the HPC system.

Except for item (3), all the others are unexpected events that are impacting the correct functioning of the FaaST Architecture and they should be taken into account, as shown in figure 9.

Figure 10 reports the execution time of the Flops Benchmark for a set of applications, grouped according to the underlying framework used: MPI, Lithops, and FaaST. The plot indicates that MPI applications achieve the shortest execution time for small problem sizes, followed by FaaST, while applications based on Lithops require the longest time to complete. Except MPI, it is noteworthy that FaaST, combined with the extendable hpc-default backend, delivers the best performance, owing to the additional workers allocated, as illustrated by the n_workers step-line in the “Scaling vs Backend” chart. Specifically, the number of workers increases from 100 to 224 after 13:20 and remains available long enough to significantly accelerate FaaST-based executions—nearly halving the runtime compared to Lithops with the hpc-baseline backend. These findings demonstrate that applications leveraging the FaaST architecture are more likely to acquire supplementary resources, thereby reducing overall execution time.

### 2.9.6 Limitations

Although many challenges were successfully addressed during the development of the FaaST architecture, several limitations persist concerning the collected data, the time-series model, and the overall design of FaaST.

- **Data collection**: The available HPC usage data are limited in granularity, aggregated across the entire system, which restricts insights into partition- or group-level behavior. Data are provided only as a univariate time series, omitting additional factors such as job-specific information from the Slurm queue. Furthermore, temporal patterns are influenced by system maintenance, outages, and user behavior variability, complicating trend analysis.
- **Time series model**: The forecasting model exhibits several limitations. While it captures general trends, prediction errors remain substantial, partly due to unmodeled seasonal components and the inherent limitations of ARIMA. Hyperparameter tuning relied on a greedy search, and model selection considered all error metrics equally rather than prioritizing those most relevant to the use case. The model also depends on stationary data and linear relationships, and it currently does not handle potential data-shift issues, which could degrade performance over time.

### 2.9.7 GWD Execution

The different integrations and architectures have been introduced in the GWD use case. These have resulted in a new implementation within the NearData project. These improvements have had a direct impact on key performance indicators, notably performance (KPI-1) and auto-scaling capabilities (KPI-3). Moreover, Lithops-HPC improvements on user experience have made GWD execution more user-friendly for non-technical users (KPI-5). Such improvements are explained in D3.2

A major advancement stems from the integration of the Lithops-HPC backend, which enables parallelized data ingestion and orchestrated analysis. This integration has led to a 36-fold increase in processing speed. Specifically, Lithops-HPC facilitates the concurrent reading of multiple files and coordinates the subsequent analytical steps, thereby eliminating redundant file access operations. Prior to this integration, the estimated computational time for data ingestion and model training was approximately 6 minutes using 6 CPUs. With the adoption of Lithops in combination with DataPlug (and the shuffle connector), execution time has been reduced to just 10 seconds under the same computational configuration. Considering a total of $15 * 10^6$ executions, this optimization enables the full execution of the GWD pipeline within a feasible timeframe of approximately 22 hours using 22,400 CPUs.

An additional advancement in the GWD use case stems from the analysis of hyperparameter optimization under combinatorial constraints. Given that genomic variation contributes to complex disease risk through predisposition rather than direct causality, we anticipated limited variability in optimal hyperparameter selection during the tuning phase. To validate this assumption and identify a robust set of hyperparameters, we conducted multiple randomized experiments using diverse variant configurations associated with disease. These experiments enabled us to cluster datasets based on their accuracy and F1-score performance metrics.

A consistent pattern emerged among the hyperparameters of the top-performing models, which were subsequently applied across datasets. Comparative analysis between fold-cross validation using dataset-specific parameters and models trained with generalized parameters revealed no statistically significant differences in performance.

This insight enabled us to define a universal set of optimal hyperparameters, effectively eliminating the need for iterative tuning. As a result, we achieved a 5-fold improvement in performance, rendering the full execution of the GWD pipeline both feasible and efficient.

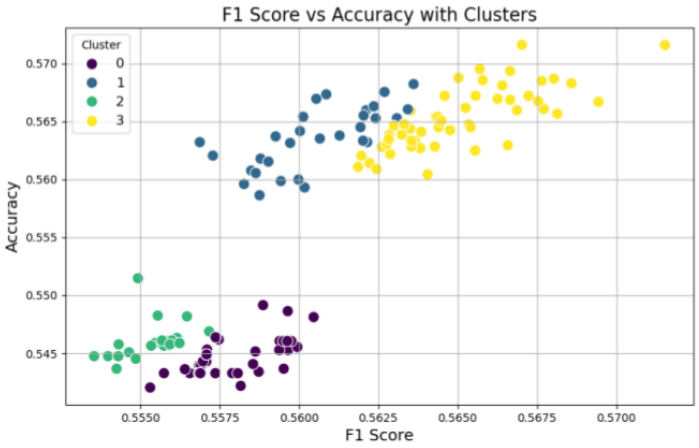



Figure 11: Model performance clustering

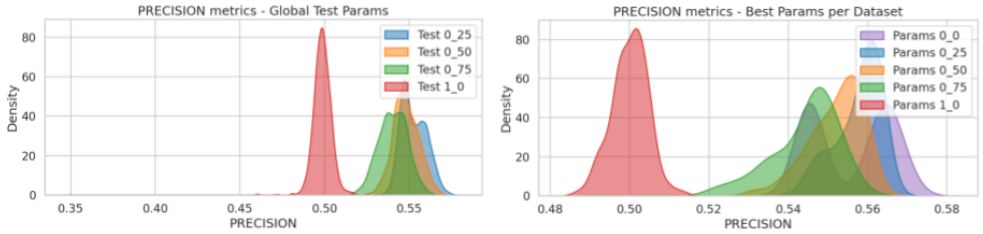



Figure 12: Parameter performance

### 2.9.8 KPIs achieved

In table 7 we summarize the KPIs achieved on this use-case with the aforementioned experiments[6]. The development of the HPC extreme data connector and its integration within Lithops, as explained in Deliverable D3.2, has allowed for speed-up improvements of up to 36x in the data ingestion of the GWD use-case, solving its main bottleneck. Moreover, further improvements on the hyperparameter tuning and selection improved its performance by another 5x. On the other hand, the Lithops-HPC data connector not only has improved performance but also the user experience by providing a much easier handling of the supercomputing resources. This enhancement allows non-technical users to manage the supercomputer and launch the GWD use-case without dealing with the hurdles of HPC resource management.

Finally, the integration of AI on resource management and its integration within the Lithops-HPC connector allows for effective scaling of resources as available in the system and enhances the execution of the use-cases by 1.5x.

## 2.10 Use-case repositories

In the next list, we indicate where to find the source code of the different connectors and the pipelines. Note that the HPC connector and ML accelerator are part of the Lithops-HPC and GWD, respectively.

[6]KPI-5 refers to metrics explained in Deliverable 3.2 as part of the HPC data connector integrated into the Lithops-HPC architecture

| KPI | Summary of results |
|---|---|
| KPI-1 | Lithops-HPC connector improves data ingestion in GWD pipeline by **36x**. |
| KPI-1 | GWD hyperparameter selection improves performance by **5x**. |
| KPI-5 | Cyclomatic-Complexity reveals **1.5x** fewer execution paths. |
| KPI-5 | Yaqin's metrics reveals **1.6x** fewer branches, loops and nesting depth. |
| KPI-3 | The auto-scaler improves execution time of MDR use-case by **1.5x**. |

Table 7: Summary of KPIs achieved on this use-case

- GWD pipeline: `https://gitlab.bsc.es/datacentric-computing/lithops-hpc-genomics`
- MDR pipeline: `https://gitlab.bsc.es/datacentric-computing/mpi-genomics-mdr`
- MDR pipeline with Lithops: `https://github.com/neardata-eu/lithops-hpc/tree/main/examples/mdr`
- Lithops-HPC: `https://github.com/neardata-eu/lithops-hpc`
- Lithops-HPC user guide: `https://gitlab.bsc.es/datacentric-computing/lithops-hpc-examples`
- Data shuffle connector: `https://github.com/danielBCN/dataplug/tree/filesystem`
- Auto-scaling connector: `https://gitlab.bsc.es/datacentric-computing/lithops_telemetry_forecasting/lithops_deployment_agent`

### 2.11 Conclusions

This use-case presented multiple challenges: (1) a bottleneck on the ingestion of the data due to an inefficient reading and huge datasets, (2) a challenge on the way the dataset was presented with a structurally ordered data in terms of variants and/or chromosomes, (3) the challenge of efficiently selecting hyperparameters of the model to identify potentially problematic variant pairs, which was computationally very expensive and (4) improving those issues while addressing user experience.

To solve these issues, we have combined Lithops, a serverless platform capable of scaling to thousands of cores in a user-friendly manner, and the supercomputing resources made available after the award of two grants to access such infrastructure. The combination resulted in an HPC connector that allowed overcoming the security issues encountered in such HPC environments and made Lithops able to run without the traditional cloud backends. With that, we improved (1) data ingestion and (4) user-friendliness.

Moreover, we have developed a data shuffle connector that not only partitions the data but also shuffles it to avoid the inherent order of the data in the original dataset, solving challenge (2). Finally (3) was solved using innovative techniques for hyperparameter selection that improved the training stage of the model by 1.5x.

Additionally, we have combined all this with an auto-scaling AI-based methodology to add resources to the computational backend as soon as they become available. This avoids workloads from starving for a long waiting period, awaiting the resources to be available. This was possible since we can now start running with as many resources as available in the system and then increase as they become more available, so overall the time to service is improved.

### 2.12 Future work

We leave as future work to improve the usage of the system's storage, as of now, it uses the centralized shared storage, which is at times inefficient. To do so, we plan to use GekkoFS system, leveraging the local hard disks of the computational nodes as a cache. Doing so, however, poses the challenge of dealing with many open network connections across the different nodes efficiently. On

the other hand, improvements can be made on the resource management of our Lithops-HPC platform, adapting state-of-the-art policies to the use-case requirements. Finally further improvements can be done on the auto-scaler. Further research will focus on refining the Time Series model for system occupancy, incorporating GPU acceleration for high-throughput workloads, and extending the approach to other computationally demanding applications. The findings indicate that FaaST's architecture provides a scalable and efficient alternative to conventional resource allocation strategies, contributing to more flexible and cost-effective computing solutions in bioinformatics and beyond.

# 3 Use Case: Computer-Assisted Video Surgery

The National Center for Tumor Diseases (NCT, Germany) [18] is an institution that mixes data scientists and surgeons to apply AI techniques on surgery-related multimedia to analyze surgical workflow with the purpose to assist surgical staff. Maier-Hein et al. have shown the importance of this research field (Surgical Data Science) which aims to improve surgical healthcare through data analysis and machine learning, with a focus on overcoming the translational challenges to implement these data-driven approaches into clinical practice [19, 20]. For this, NCT requires video data from laparoscopic cameras to be durably ingested and processed in real-time. Moreover, video data should be durably stored in long-term storage, so it can be accessed via batch analytics (*e.g.*, AI model training). In deliverable D5.1, we presented a proof-of-concept for NCT by using Pravega and GStreamer for managing video streams and feeding containerized AI jobs.

Nonetheless, a new challenging requirement for NCT is to search for specific video fragments, or even individual frames, in a collection of video streams. This is required for multiple reasons, ranging from the creation of specialized AI model training datasets to help surgeons or medical students locating specific video fragments of certain anatomies. If we consider a large collection of video streams, using just the metadata of video streams is not enough to satisfy content-based queries. Moreover, data scientists may require to find videos based on unstructured data as input, like an image. We need a content-based approach for indexing, querying, and retrieving relevant video stream data in streaming systems.

## 3.1 Short Description

While streaming systems are often associated with managing event-like data types (*e.g.*, logs, sensors), there is an increasing interest in using these systems for managing multimedia. For instance, AWS Kinesis can ingest video streams and serve them in real-time to analytics applications [21]. Similarly, Pravega [22] offers a GStreamer connector for building video analytics pipelines [23]. Use cases like object identification in surveillance cameras, computer-assisted surgery, and quality control on manufacturing processes are just some examples that motivate streaming systems to support video streams as a first-class citizen [24, 25].

Furthermore, with the advent of AI, video streams are increasingly valuable not only when processed in streaming fashion, but also in batch. For instance, historical video data is a key asset in AI-related tasks like model training [26]. We also realize that this aligns naturally with the shift of streaming systems towards supporting storage tiering for data streams [27, 28]. Tiering stream data to a scale-out, cost-effective storage is an ideal solution when ingesting and storing video data for extended periods of time.

Unfortunately, while tiering stream data is a key requirement for managing historical video data, this alone is not enough. When training AI models, data scientists require the means to query and locate video streams, or even specific video fragments, within a large pool of videos. Such queries could be related to the content of videos, which goes beyond what video metadata can express. This requires some sort of semantic video search solution for streaming systems that, to our knowledge, is not available today.

## 3.2 Challenges addressed

Our main goal is to devise a flexible semantic video search solution for streaming systems. Achieving this goal could provide added value to data streaming platforms supporting video stream ingestion and analytics [29, 30]. For example, users could index video streams based on their own models and get accurate query results in the form of video fragments. Even more, data loaders in AI inference frameworks could exploit such a mechanism by ingesting only relevant video fragments for training a model, discarding the rest.

While promising, achieving this goal entails some challenges:

*(C1) Flexible content-based video stream indexing*: Although a video stream is immutable, its contents

[6]The content of this section maps to task T5.6 and is related to the paper "*StreamSense: Policy-driven Semantic Video Search in Streaming Systems*", published in ACM/IFIP Middleware'24 (Industry Track).

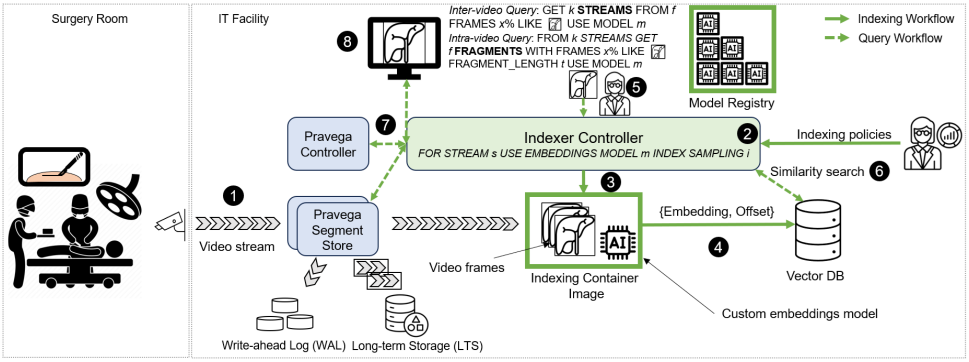



Figure 13: Overview of StreamSense design and main components in the NCT use case.

can be indexed in multiple ways. For instance, a data scientist may use a surgery video for training an AI inference model for liver segmentation, while another user may resort to the same video stream for learning surgery techniques with specific instruments. As one can infer, both users are interested in different fragments of the same video stream. But supporting queries for both users requires the system to understand what a "liver" and a "surgical instrument" are in order to build an index accordingly. Therefore, our solution should enable users to (re)index the same video stream based on different models.

*(C2) Scalable semantic search*: Dealing with tiered video streams for long retention periods requires managing large amounts of index data. In use cases like health video analytics, we may require indexing each video stream with fine granularity (*e.g.*, embedding per key video frame). On the one hand, it seems apparent that building a global index for all the videos would not scale, as the amount of required memory per query is, in some cases, the size of the indexed vectors [31, 32]. On the other hand, the cost of querying individual per-stream indexes may grow linearly with the number of video streams. Thus, we need to devise an index management approach that provides a reasonable solution to this trade-off.

*(C3) Programmatic search interface*: We see potential in exposing a semantic search mechanism via APIs to external programs, in addition to serve data scientists. For instance, AI inference frameworks could reduce data transfers when loading data that is related to the specific model to train, instead of bulk loading a whole collection of video streams. However, this topic requires further exploration.

### 3.3 Targeted KPIs

This work is related to the following project KPIs:

- **KPI-2 - Video Indexing Performance**: We evaluate on the indexing performance of StreamSense both in streaming and batch.
- **KPI-2 - Semantic Video Search Latency**: We analyze the latency of inter/intra video queries in StreamSense.
- **KPI-1 - Data Transfer Savings in AI Data Loading**: We analyze the data ingestion problem when data scientists attempt to download relevant video data for training their AI models.

### 3.4 Integration with NEARDATA architecture components

Next, we describe the design of StreamSense: our policy-driven semantic video search solution for streaming systems. Moreover, we describe its integration with other NEARDATA components.

#### 3.4.1 Policy-driven Video Embeddings Generation

In Fig. 13, we provide a high-level overview of the architecture of StreamSense and the use case that motivates it. First, we identify the video ingestion phase. As visible in step ① of Fig. 13, video frames from surgery cameras are ingested in Pravega as *video streams*. Pravega achieves low latency

and durability upon video ingestion thanks to the WAL [33], which is expected for streaming computations. Moreover, the storage tiering mechanism in Pravega automatically moves video data to LTS. Upon a batch job processing video data, Pravega will exploit the high read throughput and parallelism of LTS systems (*e.g.*, NFS, AWS S3, HDFS). Therefore, using a tiered streaming system like Pravega achieves a sweet spot in the latency vs throughput trade-off when ingesting video data.

The *indexer controller* is the main component of StreamSense. One of its tasks is to orchestrate indexing activities via *policies* [34]. To this end, users can define policies for indexing video streams (step ② in Fig. 13). Such policies are expressive enough for capturing relevant aspects of the indexing process, like the embedding model to be used or the index sampling granularity. For example:

```
FOR STREAM s USE EMBEDDINGS MODEL m INDEX SAMPLING i
```

With the policy above, the data scientist instructs the system to index a video stream *s* via the embeddings model *m*, as well as to use an index sampling algorithm *i* (see §**??**). The indexing controller keeps the information of policies in a metadata store.

Indexing policies may be in two states: *active* and *completed*, depending on whether the indexing process for a video stream has finished or not. The indexer controller triggers the embedding generation process for active policies. Retaking the previous example, the indexer controller spawns a new *indexing instance* (*e.g.*, container or VM) with the requested embedding model running inside and consuming data from stream *s* (step ③ in Fig. 13). Note that policies apply to both real time and historical video streams. Exploiting tiered data streams abstracts stream readers from the data location, while achieving good performance in both streaming and batch computations. With StreamSense, a data scientist could trigger multiple indexing instances for the same video stream, thus generating multiple semantic indexes for the same content.

For simplicity, StreamSense provides an *indexing container image* that handles IO with the streaming system and contains the necessary dependencies. StreamSense creates a new indexing instance using the indexing container image with the appropriate model from the *model registry* (*i.e.*, repository of embeddings models on top the indexing container image). Under the hood, the indexing container image ingests the video frames, passes them through the model, and writes the index data in the vector DB.

### 3.5 Data connectors developed

The StreamSense *indexer controller is the connector* proposed in this piece of research. It deploys NCT AI models to generate embeddings from video frames and is in charge of replying to semantic queries. In what follows, we justify its design and main novelties.

In health use-cases like NCT, every frame may contain critical information pertaining to surgical events, such as serious complications. Missing video frames could result in overlooking vital details, which motivates us to store full video stream indexes (step ④ in Fig. 13). However, while vector DBs can create indexes over large vector embeddings collections, performing efficient queries on a single index containing all the video embeddings can be challenging [35]. For example, vector DBs like Milvus [36] are limited in their query capabilities to the amount of available memory [31]. On the other hand, just creating an index per video stream and embeddings model can be problematic too. To wit, since each query can only access one video stream index, the cost of searching for content may grow in the order of the number of video stream indexes. This is an interesting trade-off that calls for a vector DB-agnostic solution.

In StreamSense, we build a two-level video stream indexing layout per embeddings model (see Fig. 14). First, indexing policies define an *indexing sampling algorithm*. StreamSense currently offers time-based index sampling (*e.g.*, index a frame every 1 or 30 minutes), but more sophisticated "shot boundary" algorithms can be added in the future [37, 38]. New video sampling algorithms just need to implement the `do_sampling(frame, embedding, offset)` method in the indexing container image. The index sampling algorithm determines the embeddings that will be stored in the *stream sampling index* (via the `add_to_sampling_index(embedding, stream)` call). Such an index will contain a

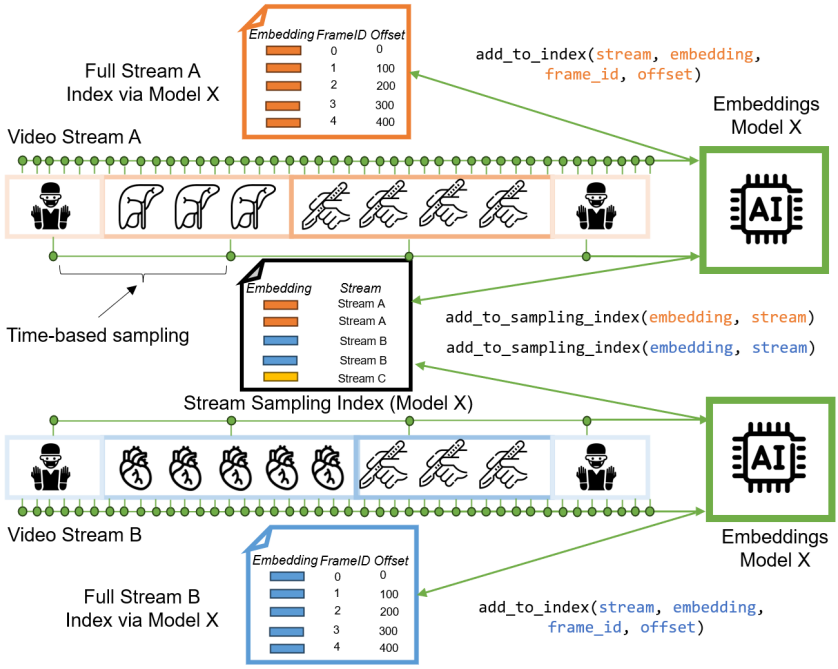



Figure 14: The stream sampling index helps us to find video streams in which any of their computed sample embeddings matches a similarity query. A per-stream index allows us running semantic search within video streams.

small set of embeddings for all the videos in the system associated to the video stream. The stream sampling index enables StreamSense to search for videos in which *any* (sampled) frame satisfies a similarity query.

An indexing instance also builds a *full index* for the stream at hand (*e.g.*, one embedding per key frame). The outcome of the embeddings model running in the indexing instance is written to the vector DB via the `add_to_index(stream, embedding, frame_id, offset)` call. With this index, StreamSense can perform fine-grained search on any video stream. This is key for finding relevant video fragments given a specific similarity criterion.

In summary, our two-level video indexing layout trades-off a small storage penalty building the sampling index (*e.g.*, 1% to 5% additional vector embeddings depending on the sampling model) for better query scalability (see results in §3.7.3). Next, we describe how StreamSense uses this index for serving semantic queries.

### 3.5.1 Semantic Video Search

The indexer controller also takes care of orchestrating the query workflow in StreamSense. First, it receives the input data for the query at hand. We consider two types of input data: *vector embeddings* and *images*. In the former, the provided embedding is directly used to perform the similarity search against the vector DB. For the latter, the indexer controller should first generate the embedding from the input image and then perform the similarity search.

Based on the generated indexes, StreamSense allows users to perform two types of queries: *inter-video* and *intra-video* (step ⑤ in Fig. 13). Queries target the index generated from the embeddings model *m* (step ⑥ in Fig. 13). Let's see an inter-video query example:

```
GET k STREAMS FROM f FRAMES x% LIKE [IMAGE] USE EMBEDDINGS MODEL m
```

Inter-video queries use the sampling embeddings index for the given embeddings model and enable users finding video streams in which any (sampled) frame matches the similarity criterion. Inter-video queries expose four parameters to users: i) the maximum number of video streams to

return to the user ($k$), ii) the number of closest sample frames to retrieve from the vector DB ($f$), iii) the similarity threshold or percentage required for result video streams ($x$), and iv) the embeddings model ($m$). The query workflow boils down to retrieving from the sampling embeddings index for model $m$ the top $k \cdot f$ sample frame embeddings most similar to the input embedding. We filter out the embeddings that do not meet the similarity threshold $x$ for retrieving the relevant best $k$ streams.

Intra-video queries allow users finding fragments within a video stream that match the similarity criterion. Let's see a query example:

```
FROM k STREAMS GET f FRAGMENTS WITH FRAMES x%
LIKE [IMAGE] FRAGMENT_LENGTH t USE EMBEDDINGS MODEL m
```

The query workflow works in two steps: first, the indexer controller runs an inter-video query with the input embedding to retrieve the most similar $k$ video stream candidates and discarding the ones not respecting the similarity threshold $x$. Then, with the resulting video stream collection, the indexer controller will search the specific video stream full indexes for the $f$ most similar frames — described by `<embedding:frame_id:offset>` triples— to the input image reference. For each video stream, the query filters the `<embedding:frame_id:offset>` triples with an accuracy that meets the embedding similarity threshold $x$. The length of the video fragment surrounding a relevant frame is determined by the query parameter $t$ (*e.g.*, 20 seconds). The cost in terms of vector DB queries of this process is 1 query to the sampling embeddings index and up to $k$ queries to find frames to the individual video stream full indexes, which can be parallelized for better performance. Note that it may be the case for several relevant frames to be consecutive in a video stream. To minimize redundant results, video fragments with overlapping offsets are merged into a single one.

Both inter/intra-video queries are exposed via APIs in the indexer controller (*i.e.*, `find_streams(image, k, f, x, m)` and `find_fragments(image, k, f, x, t, m)`). This enables external programs to exploit the semantic video search in StreamSense. We believe the use-case of collecting relevant datasets for AI training in inference frameworks (*e.g.*, PyTorch) is especially interesting. As we assess in §3.7.4, this allows users to easily find and load relevant training datasets, while reducing data transfers during the process.

Finally, the indexer controller gets the video stream names or video fragments offsets from a query and interacts with Pravega for displaying the query results back to the data scientist (step ⑦ in Fig. 13). StreamSense also provides facilities for visualizing the outcomes of queries, thus providing a full end-to-end solution for semantic video search (step ⑧ in Fig. 13).

### 3.6 How AI is enabled in the use-case

AI is central to the NCT surgery use case through the StreamSense framework, which enables semantic indexing and search of surgical video streams:

- **Leverage deep learning models** (*e.g.*, ResNet50) generate semantic embeddings from surgical video frames.
- StreamSense uses these **embeddings to build searchable indexes** for real-time and batch video analytics.
- Two-level indexing enables **scalable inter- and intra-video semantic search**.
- **APIs allow querying with images or embeddings** to retrieve relevant video fragments.
- **Integration with PyTorch** enables AI model training using only semantically relevant data, reducing transfer overhead.

### 3.7 Results and KPIs achieved

Our evaluation focuses on: i) what is the video indexing performance of StreamSense? (§3.7.2), ii) what is the semantic search latency of video streams/fragments? (§3.7.3), iii) can StreamSense reduce data loading phase for AI training via semantic search? (§3.7.4).

### 3.7.1 Setup

We summarize here the configuration used in our AWS experiments.

*Prototype Implementation*. Our implementation relies on Pravega for video stream IO via the GStreamer connector [23] and Milvus [36, 39] as a vector DB. We also have a base docker image with the necessary dependencies for managing video streams from Pravega which are consumed by AI inference models. The indexer controller is implemented in Python and manages indexing policies and the life-cycle of indexing instances (*e.g*, as VMs or Kubernetes pods). The code StreamSense is publicly available [40].

*Deployment*. StreamSense is deployed in a EC2 cluster of 4 nodes (`us-east-1a`) inside the same VPC. Pravega uses an `i3en.2xlarge` instance (8 vCPUs, 64GB of RAM, and 2 NVMe) running a Pravega controller, a Pravega segment store, a Bookkeeper instance, and a Zookeeper instance. We use an EFS share as long-term storage for Pravega. The indexer controller runs on a `p3.2xlarge` instance (8 vCPUs, 61GB of RAM, and an NVIDIA V100 GPU) jointly with the AI inference model. An official standalone Milvus hosted on a `m5.2xlarge` instance (8 vCPUs, 32GB of RAM) configured with a global `Collection` that will store the embeddings from the sampled frames, and a `Collection` per video stream that will store the embeddings from all the key frames (*e.g.*, 1 per second). In Milvus, we use the `Inverted File Index` index type with 64-point clusters and the cosine similarity metric for the search and `bounded staleness` consistency mode (default). A `c5.4xlarge` VM (16 vCPUs, 32GB of RAM) generates `h264` (25FPS, 940x560) video streams emulating surgery cameras. We run a GStreamer pipeline that reads MP4 files from our datasets and writes them to Pravega as streams.

*Datasets and Embedding Models*. We utilize three publicly available datasets of surgical videos: CATARACTS [41], which includes videos of cataract surgeries; CHOLEC80 [42], which comprises videos of cholecystectomy surgeries; and AUTOLAPARO [43], which consists of videos of laparoscopic hysterectomy surgeries. To extract the embeddings, we make use of ResNet50 [44] trained with the `IMAGENET1K_V1` dataset for its performance in our similarity use case. It is also widely used for surgery processes, as seen in [45, 46].

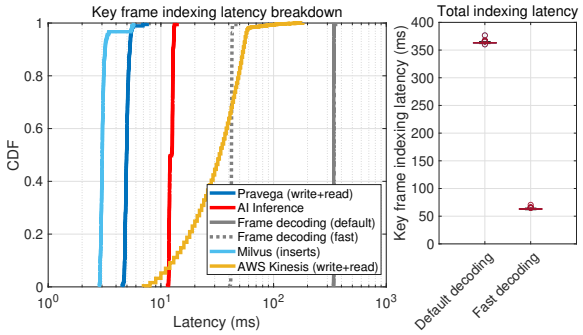
### 3.7.2 Video Indexing Performance

Next, we focus on the indexing performance of StreamSense both in streaming and batch (Fig. 15). In the streaming case, Fig. 15a the total indexing latency for key video frames and the latency breakdown. Indexing latency is measured by time-stamping video frames at the source and calculating the delta through the indexing process.

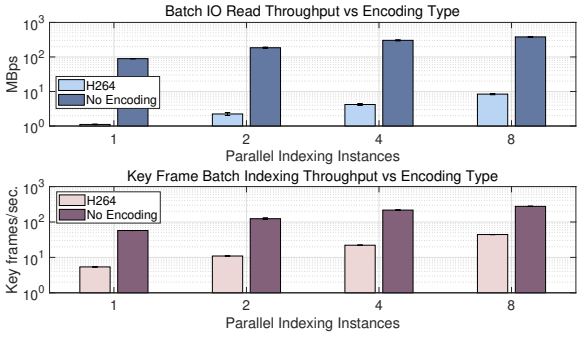
In Fig. 15a, the total key frame indexing latency ranges between 63ms and 360ms in average, depending on the decoding configuration. This is the waiting time for a data scientist to perform semantic search on ingested video frames in StreamSense. We observe that video decoding is a compute-intensive task that seems to dominate indexing latency. A plausible explanation to this is that, while inference models use the available VM GPU, GStreamer decoding by default uses CPU. Some decoding configurations also entail buffering frames, which has a toll on latency as well. The latency difference between the default GStreamer `decodebin` and the `fast` configuration (*i.e.*, `ultrafast` speed preset, `fastdecode`) gives a sense on the room for optimization in this indexing stage.

Fig. 15a shows a latency breakdown indexing key frames in StreamSense. The Pravega IO latency shows a p99 latency under 7ms. This confirms that Pravega achieves good performance for managing video data in real-time, while providing automated storage tiering to LTS. For comparison, we have performed a micro-benchmark measuring the latency of writing and reading 10KB events in AWS Kinesis at a rate of 25 events/second (same rate as our video streams). Visibly, Kinesis exhibits a higher IO latency than Pravega (*e.g.*, 10.7x higher latency at p95) and does not provide storage tiering. The AI inference latency to generate embeddings takes around 12-14ms per key frame and inserts to Milvus take 3-6ms. This latency meets the requirements for streaming workloads. Again, the video decoding phase dominates latency (30-330ms).

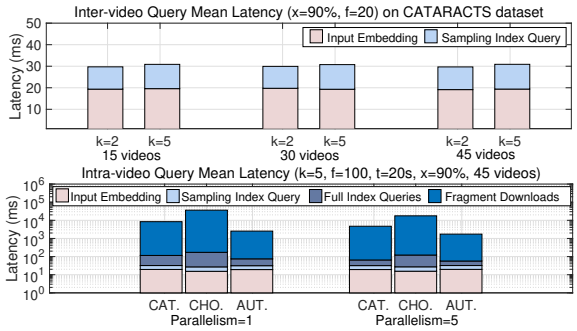
Next, we focus on the batch indexing throughput depending on the video encoding configuration. Fig. 15b compares the indexing throughput of parallel VMs depending on if video uses `h264` encoding (`fast` configuration) or no encoding (*i.e.*, raw frames). Visibly, there is an interesting trade-off




(a) Key frame indexing latency.




(b) Batch key frame indexing throughput.




(c) Semantic query latency.

Figure 15: StreamSense performance: i) key frame streaming indexing latency with `h264` encoding (left), ii) throughput indexing parallel streams in batch depending on the encoding type (center), and iii) inter/intra stream semantic query latency (right).

between indexing throughput and data storage/network efficiency. With `h264` encoding, VMs index video at a rate of 8.5MBps after decoding (5.5 key frames/second), which translates to $\approx$ 1MBps of read throughput ($8x$ transfer reduction). On the other hand, an indexing VM with the `no_encoding` option achieves an indexing throughput of $\approx$ 55 key frames/second thanks to avoiding the decoding overhead, which translates into 85MBps at the IO level. In the 8-VM case, the Pravega instance is saturated at 400MBps. However, the raw video storage size is around 200x larger compared to `h264` encoded video. StreamSense lets users to pick the best encoding based on their needs.

### 3.7.3 Semantic Video Search Latency

Next, we evaluate the latency of inter/intra video queries in StreamSense. The upper plot in Fig. 15c shows the latency of querying the stream sampling index to find streams ($k = 2$, $k = 5$) whose sampled frames meet a certain similarity threshold ($x = 90\%$) for various dataset sizes. Visibly, neither the tested number of streams to retrieve ($k$) nor the tested dataset sizes (15GB to 50GB) seem to have an impact on mean query latency, which is around 30ms. Most of the inter-video query time is spent on generating the input image embedding ($\approx$ 19ms), whereas $3-4$ms are related to Milvus query and the rest to filtering query results. This is mainly because the sampling index grows slowly, as the sampling interval is set to 30 seconds. We have micro-benchmarked (VectorDBBench [47]) our Milvus instance and found that the sampling index exhibits similarity search latency $>$ 15ms with 0.5 million embeddings. With the same sampling interval, that would allow us storing 4.2k video hours before sampling index queries take two digit milliseconds.

The lower plot in Fig. 15c shows the intra-query latency per dataset downloading $f = 100$ video fragments of $t = 20$ seconds from the top $k = 5$ video streams that meet a $x = 90\%$ similarity. As expected, depending on the dataset, $>$ 98% of query time is related to downloading video fragments from Pravega. This is reasonable as the amount of data to be downloaded ranges from 6MBs to

| 15 *videos* | CATARCT query | CHOLEC query | LAPARO query |
|---|---|---|---|
| $k = 2, f = 50$ | 89.12% | 97.13% | 99.77% |
| $k = 5, f = 100$ | 83.79% | 87.97% | 99.77% |
| 45 *videos* | CATARCT query | CHOLEC query | LAPARO query |
| $k = 2, f = 50$ | 97.00% | 98.38% | 99.83% |
| $k = 5, f = 100$ | 93.94% | 92.82% | 99.79% |

Table 8: Data transfer reduction of downloading relevant intra-video query fragments vs transferring full dataset.

160MBs. Still, we observe that increasing query parallelism from 1 to 5 threads reduces intra-video query execution in 32% to 51%.

#### 3.7.4 Data Transfer Savings in AI Data Loading

In this experiment, we address the problem of data scientists downloading relevant video data for training their AI models. In Table 8, we exercise the preliminary integration our PyTorch data loader for StreamSense. We allow the data loader to use an embedding as input —as well as the other intra-video query parameters— and calling the StreamSense service for downloading similar video fragments. As can be observed, StreamSense allows data scientists to automatically load relevant data for training their AI models while achieving a transfer reduction from 83.79% up to 99.83% compared to bulk transferring the whole dataset, depending on the intra-video query parameters ($k$ and $f$) and the dataset.

#### 3.7.5 Evaluation of Video Loading Strategies in NCT

We also performed a battery of experiments exploring a data loading connector for Pravega and GStreamer. To wit, in this project have shown that the utilization of Pravega and GStreamer to store laparoscopic video streams within the framework offers compelling advantages in comparison to typical data storage of images or videos on network storages, particularly in the context of high-volume, real-time surgical video data. Furthermore, the framework is able to handle continuous, high-resolution data streams efficiently and addresses the extreme data challenges inherent in modern surgical environments. In this use case we do not only need to capture and process videostreams with pravega but we also want to load also Pravega-GStreamer-Videostreams to perform machine learning to train new models. Therefore, the research question of how much can we increase our data loading speed with Pravega-GStreamer compared to traditional methods of storing and accessing surgical video data was tackled.

Our experimental setup was designed to compare the performance of different data loading strategies for surgical video data. We utilized three distinct dataloaders:

- **Frame Dataloader:** This method involves pre-extracting individual frames from videos and storing them, then loading these stored frames for processing.
- **Pravega Dataloader:** This approach leverages Pravega to stream and load frames directly from a Pravega stream.
- **Video Dataloader:** This involves loading frames directly from a Ceph-based network-attached storage (NAS).

As an example, we used a 16-minute long video as the source data. To simulate data loading during Machine Learning (ML) Training, frames were loaded at a rate of one frame every two minutes. Loading frames for ML three different processes happens: preparation, initialization, and frame loading. The first two are executed once and the last one for every batch. The time needed for preparation is the time for extracting frames, once before using the data is possible. The initialization time is the

time needed per video when initializing the dataloader, e.g. counting frames and extracting frames from the video stream. The loading time is the time to load a single frame for training. The following table shows the time needed to prepare the data, initialize the dataloader, and load individual frames.

Table 9: Data Loading Strategy Performance Comparison

| Time (s) | Preparation Time | Initialization Time (s) | Loading Time (s) | Storage on Disk |
|---|---|---|---|---|
| Frames | 0.336746 ± 0.025617 | 0.000095 ± 0.000023 | 0.003397 ± 0.000457 | large |
| Pravega | 0 (assuming videos are already in pravega as they are streamed there during the OR to enable assistance functions) | 11.995750 ± 0.325696 | 0.018268 ± 0.007544 | small |
| Video | 0 (assuming videos are directly stored on the network storage) | 0.008763 ± 0.002081 | 0.027855 ± 0.002863 | small |

The expected change when the video length grows is displayed in Table 2. For the frame data loader a significant increase of storage on the disk is needed for extracted frames. The Pravega data loader with its current setup needs more time to get corresponding timestamps to load from Pravega, whereas the video data loader the loading time takes more time because with longer videos more frames need to be loaded.

The analysis of the results presented in Table 1 and Table 2 highlights the strengths and weaknesses of each data loading strategy, particularly concerning their performance under varying video lengths.

The Frame Dataloader demonstrates the fastest initialization and loading times for individual frames, making it efficient for immediate access once data is prepared. However, its significant preparation time (for extracting frames) and the large to significant increase in storage on disk with growing video length pose considerable practical limitations for high-volume, real-time surgical video data. This method essentially trades storage efficiency and initial setup time for rapid individual frame access.

The Pravega Dataloader exhibits a unique set of advantages. Its preparation time is negligible (0 seconds), primarily because it assumes videos are already streamed into Pravega for intraoperative analysis jobs, suggesting excellent synergy with existing clinical workflows. While its initialization time is the highest among the three, and increases with video length, though loading time remains stable, its small storage footprint on disk is a major benefit for handling large datasets. The current implementation's higher initialization time for getting corresponding timestamps from Pravega presents an area for future optimization, but overall, it offers a good balance between data accessibility and storage efficiency for continuous streams.

The Video Dataloader offers the simplest setup with zero preparation time and minimal initialization time, assuming direct storage on network-attached storage (NAS). Its storage on disk is "small" and increases proportionally with video length. However, its "Loading Time" is the slowest, and this

Table 10: Comparison of data loading strategies: Symbols indicate expected change with increasing video length: ^ for increase, ^^ for significant increase, and = for no significant change.

| | Preparation Time | Initialization Time | Loading Time | Storage on disk |
|---|---|---|---|---|
| Frames | ^ | = | = | ^^ |
| Pravega | = | ^ | = | = |
| Video | = | = | ^ | ^ |

time "increases" significantly with longer videos, as more frames need to be loaded directly from the video file. This method is straightforward but less performant for intensive, real-time ML training, where rapid frame access is critical.

In conclusion, the Pravega Dataloader appears to be a good trade-off between the other two dataloaders for the specific use case of machine learning training with surgical video data. Its ability to integrate seamlessly with real-time video streams, assuming existing Pravega infrastructure from OR assistance functions, and its efficient storage make it suitable for extreme data challenges in modern surgical environments. While improvements in its initialization time are desirable, its overall performance and synergy with clinical data capture workflows position it as a robust solution for enabling AI in surgery. The study effectively demonstrates the compelling advantages of utilizing Pravega and GStreamer for handling continuous, high-resolution data streams in comparison to traditional data storage methods.

### 3.8 Use-case repositories

For this work, we used three publicly available datasets of surgical videos (see §3.7). The code for StreamSense is publicly available at [40].

### 3.9 Data Spaces

Throughout this project, our primary focus will be on utilizing existing dataset to thoroughly test and fine-tune the developed systems. Below are some datasets currently available for our use:

1. Dresden Surgical Anatomy Dataset (DSAD) (University Hospital Dresden and NCT/DKFZ) [3]
2. HeiChole (University Hospital Heidelberg and NCT/DKFZ) [4]
3. Cholec80 (University of Strasbourg) [5]
4. Synthetic video data (NCT/DKFZ) [5]
5. GLENDA
6. SurgicalActions160
7. hSDB-instrument
8. LapGyn4
9. ESAD
10. PSI-AVA

Furthermore, NCT's close collaboration with the surgical facilities of the university hospital campus will enable the expansion of our dataset collection to encompass a broader spectrum of surgical workflows. As an example, within the scope of this project, the Appendix300 dataset has been compiled. This dataset comprises intraoperative minimally invasive appendectomy videos from multiple German hospitals, representing a diverse patient demographic.

### 3.10 Discussion and final remarks

The validation of the NEARDATA platform in the NCT surgery use case demonstrates the feasibility and effectiveness of integrating AI-driven semantic search into streaming architectures for clinical video analytics. The StreamSense framework enables flexible, policy-driven indexing of surgical video streams and supports both real-time inference and batch-oriented training workflows.

The system successfully addresses three key project KPIs:

- KPI-2 – Video Indexing Performance: StreamSense achieves low-latency indexing of key video frames, with total latency ranging between 63ms and 360ms depending on decoding configuration. This confirms its suitability for real-time workloads.

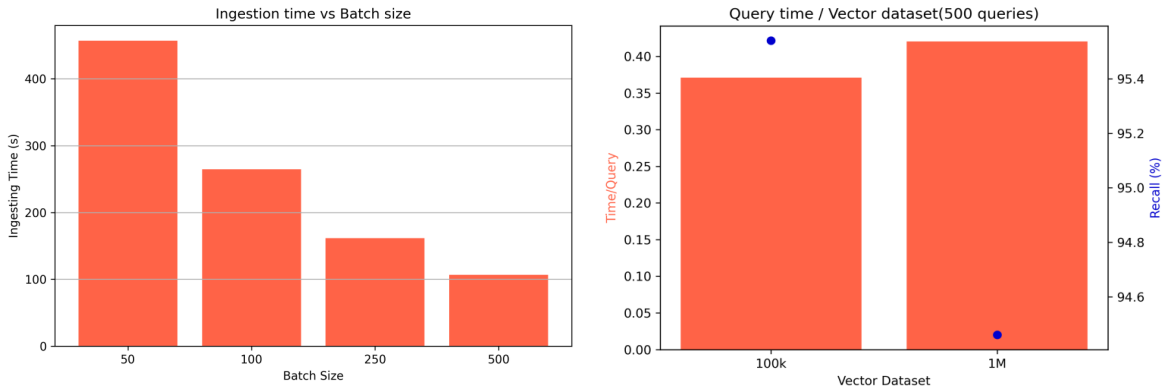



Figure 16: Preliminary data ingestion and query performance results for our S3 vector API measurement.

- KPI-2 – Semantic Video Search Latency: Inter-video queries consistently achieve mean latencies around 30ms, while intra-video queries benefit from parallel execution, reducing latency by up to 51%.
- KPI-1 – Data Transfer Savings in AI Data Loading: Integration with PyTorch data loaders enables targeted retrieval of relevant video fragments, achieving data transfer reductions from 83.79% to 99.83% compared to bulk dataset loading.

The two-level indexing strategy—combining a global sampling index with per-stream full indexes—proved effective in balancing scalability and precision. This design enables StreamSense to scale to large video collections while maintaining fine-grained search capabilities.

### 3.11 Future work

This work shed light on the potential applicability of streaming storage and AI embedding models for building a semantic video search platform in NCT. In this regard, a key element is the vector database (DB) used for storing and querying video frame embeddings. There is an increasing interest in the convergence of vector DBs and object storage, which clearly aligns with NEARDATA's approach and research lines. For example, AWS has recently launched S3 vectors[7], which is a new API that allows managing vector embeddings directly from an S3 bucket. As visible in Fig. 16, we have recently performed a measurement analysis of this s3 vectors to evaluate the practicality of object storage-based vector DBs for use cases like NCT. Moreover, this directly relates to our research regarding serverless vector DBs on top of object storage reported in D3.2. This new branch of research will help us to understand the trade-offs and practicality of vector DBs beyond typical stateful services.

[7] `https://aws.amazon.com/en/s3/features/vectors`

# 4 Use Case: Computer-Assisted Surgery - Federated Learning

## 4.1 Short Description

In the previous part of the surgical use case we investigated how to search for specific video fragments, or individual frames, in a collection of video streams. With this we can get data to train machine learning models. For the surgical use case we need advanced models who perform robust and well on heterogeneous data to minimize false prediction. A key focus is on leveraging foundation models, which are defined as large-scale, pre-trained models capable of performing a wide range of tasks and adapting to new domains with minimal data. However, the development of these models is extremely data-intensive. We can employ two main data strategies. One approach involves using publicly available images for training. While easily accessible, these images might not fully represent the specific domain of surgical procedures. Alternatively, we can utilize unpublished hospital data from platforms such as Pravega, GStreamer, or Orpheus. This second method offers the benefit of highly relevant data. Nevertheless, the aggregation of data from multiple hospitals is restricted by significant data-sharing restrictions due to regulations such as GDPR [48] and HIPAA [49]. To overcome this, we adopted the federated learning paradigm for our second use case experiment. Federated learning is a decentralised machine learning approach that trains models across multiple distributed servers holding local data samples, without exchanging the data itself. In collaboration with TUD Dresden University of Technology, we ensured the secure and protected execution of our federated learning framework using SCONE. Furthermore, we organized a EndoViS sub-challenge Federated Learning for Surgical Vision (FedSurg) 2024. The EndoViS challenge format is an international benchmarking for computer vision algorithms in endoscopic video analysis, to evaluate current strategies in the Surgical Data Science domain. The goal of our sub-challenge is to benchmark the current state of the art of federated learning strategies on a newly collected Appendix300 dataset [50]. In particular, to test the generalization and adaptation ability of the models trained within the federated setup. Finally, besides this, another experiment aimed to demonstrate that federated learning performs comparably to, or even outperforms, traditional centralized learning, thereby proving its viability as a secure and effective solution for training foundation models on sensitive surgical data.

## 4.2 Data Spaces

For the FedSurg sub-challenge we used a preliminary version of the Appendix300 dataset [50]. This dataset is curated through our close collaboration with the surgical facilities of the university hospital campus and their connection to other hospitals world wide and our goal of expanding our dataset collection to encompass a broader spectrum of surgical workflows.

The Appendix300 dataset addresses the critical need for diverse and representative training data in surgical AI by providing 330 laparoscopic video recordings of appendectomies and control surgeries, collected from five German medical centers, along with detailed clinical metadata and annotations of intraoperative appendicitis grades [50]. This comprehensive, multicentric dataset includes patient demographics, medical history, clinical symptoms, laboratory parameters, and histopathological findings, facilitating novel computer vision validation tasks in laparoscopic surgery, such as classifying appendicitis grades and differentiating perforated from non-perforated cases. Despite some limitations, including incomplete clinical data for certain parameters and the inherent ambiguities of the Gomes et al. [51] classification used for grading, Appendix300 represents a significant step forward in developing clinically relevant AI-based surgical video analysis tools and enables the evaluation of decentralized machine learning approaches.

Otherwise, we focused on utilizing existing dataset to thoroughly test and fine-tune the developed systems. Below are some datasets currently available for our use:

- Dresden Surgical Anatomy Dataset (DSAD) (University Hospital Dresden and NCT/DKFZ) [52]
- HeiCo and HeiChole (University Hospital Heidelberg and NCT/DKFZ) [53, 54]

- Cholec80 (University of Strasbourg) [55]
- Synthetic video data (NCT/DKFZ) [56]
- GLENDA [57]
- SurgicalActions160 [58]
- hSDB-instrument [59]
- LapGyn4 [60]
- ESAD [61]
- PSI-AVA [62]

### 4.3 How AI is enabled in the use-case

AI is integral to the surgical use case as we develop and utilize AI models to address challenges in Federated Learning and surgical video streaming. We aim to create robust and well-performing ML models, particularly foundation models, for tasks like video appendicitis stage classification, semantic segmentation, and classification. Our work involves leveraging federated learning to overcome data-sharing restrictions and demonstrating its viability compared to centralized learning for training these models. Furthermore, we integrate AI inference models into surgical video streaming applications to provide real-time assistance to surgeons via specialized GStreamer plugins for segmentation, phase detection, and tool detection.

### 4.4 Integration with NEARDATA architecture components

Our use case leverages two key components of the NearData architecture: the data plane component Pravega for efficient data streaming and storage, and the control plane component Scone to ensure secure and confidential execution of our experiments.

### 4.5 Targeted KPIs

Our targeted KPIs are a mixture out of following main KPIs:

- KPI-2: Significant data speed improvements (throughput, latency) in storage and access, and real-time analysis of video and sensor data.
- KPI-3: Demonstrated resource auto-scaling for batch and stream data processing, validated thanks to data-driven orchestration of massive workflows.
- KPI-4: High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.
- KPI-5: Demonstrated simplicity and productivity of the software platform, validated with real user communities in International Health Data Spaces.

Within this Federated Learning experiment, we are tackling following KPIs:

- **KPI - FedSurg Sub-challenge:** Evaluate the generalization and adaptation performance of current state-of-the-art federated learning methods in surgical vision, specifically for appendicitis classification.
- **KPI - FL-EndoViT Foundation Model:** Demonstrate the viability of using federated learning to train a robust and generalizable foundation model, validating its performance on several surgical downstream tasks.
- **KPI - Secure Computing and Platform Validation:** Validate the security and efficiency of the federated learning framework by confirming that SCONE provides a secure and high-performing environment for model training on confidential data while maintaining platform simplicity and productivity.

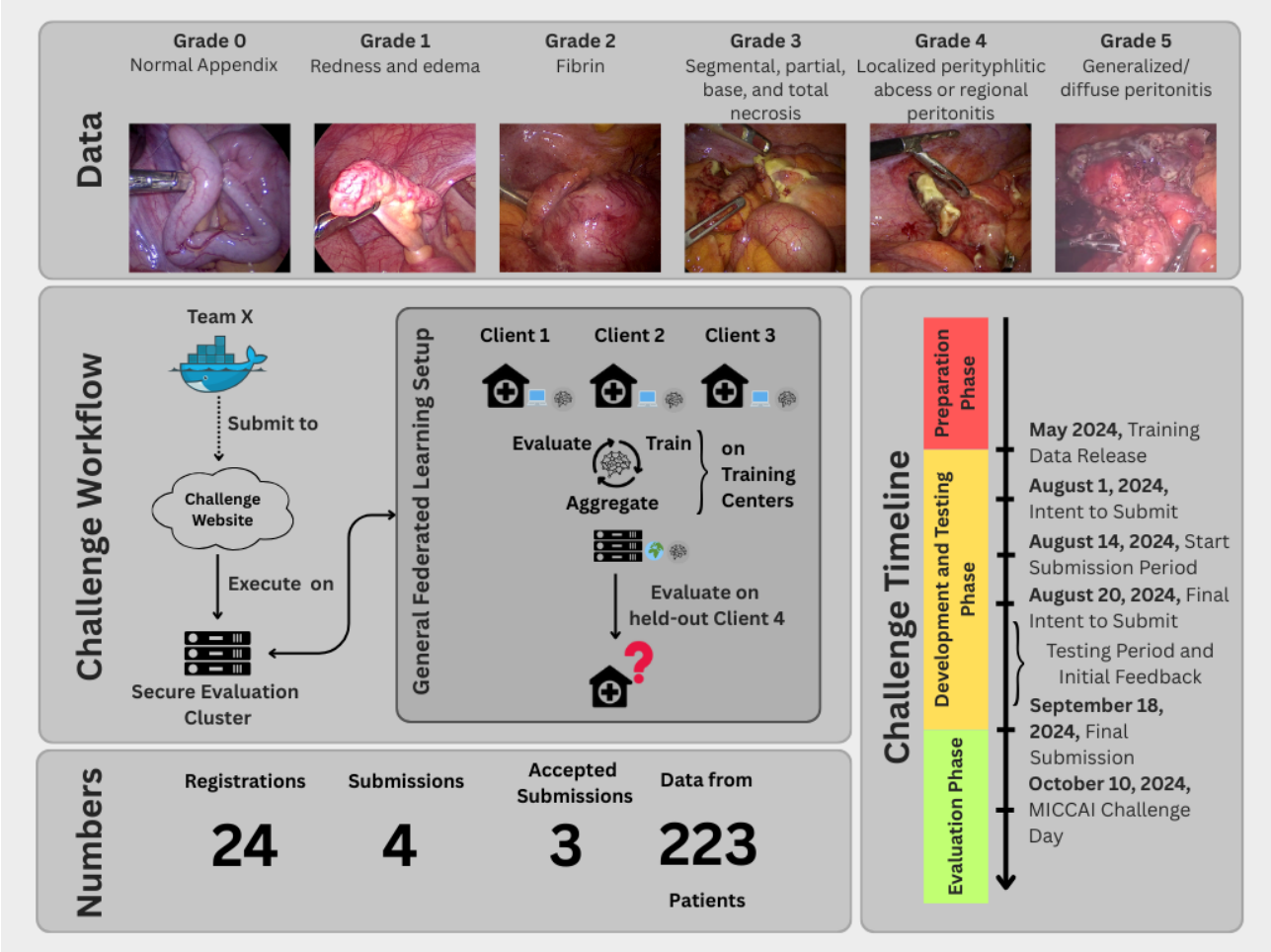



Figure 17: **FedSurg24 Challenge Highlights:** The top panel shows example images of intraoperative appendicitis grades, defined according to Gomes et al. [51], which were used for video annotation. The lower panel illustrates the FedSurg Challenge workflow: teams submitted Docker containers via Synapse, which were executed on a secure cluster simulating FL across three centers with local training and centralized aggregation. Final performance was assessed by testing each center's best local model on its own test set, while the global model was evaluated on the unseen hold-out center to measure generalization. The challenge timeline with key dates is shown alongside.

## 4.6 Results and KPIs achieved

### 4.6.1 FedSurg Challenge

FedSurg24 Challenge Highlights: The top panel shows example images of intraoperative appendicitis grades, defined according to Gomes et al. [51], which were used for video annotation. The lower panel illustrates the FedSurg Challenge workflow: teams submitted Docker containers via Synapse, which were executed on a secure cluster simulating FL across three centers with local training and centralized aggregation. Final performance was assessed by testing each center's best local model on its own test set, while the global model was evaluated on the unseen hold-out center to measure generalization. The challenge timeline with key dates is shown alongside.

The primary objective of the challenge was to benchmark various Federated Learning strategies within the domain of surgical vision, with a specific focus on examining the inherent trade-off between personalization and generalization that arises from data heterogeneity across the centers. The challenge utilized a subset of the Appendix300 dataset [50], comprising laparoscopic video recordings from appendectomy procedures conducted at four distinct German medical centers. Challenge

participants developed FL algorithms to classify appendicitis stages. The evaluation focused on two principal goals: first, assessing the global server model's ability to generalize to data from an unseen medical center, thereby demonstrating the clinical benefit of utilizing models from other centers. Second, it aimed to evaluate the local client model's capacity for adaptation, specifically its ability to be fine-tuned and perform effectively on data from each individual training center. This dual approach incentivizes clinical institutions to participate in federated learning initiatives by providing them with high-performing models for practical application, ultimately encouraging the development of robust, adaptable, and privacy-preserving artificial intelligence models across diverse surgical environments. Evaluation was based on expected cost (EC) and F1-Score metrics, with bootstrapping employed to ensure ranking stability. Additionally, a Wilcoxon signed-rank test is conducted to confirm the statistical significance of the ranking. Among the four received submissions, three teams presented valid methods. Team Santhi utilized a pre-trained Video Vision Transformer (ViViT) [63] with a frozen backbone, choosing to fine-tune only the final classification layer. Their approach used FedAvg [64] for model aggregation within the Flower FL framework and placed emphasis on specific temporal regions within the video frames. Team Elbflorenz adopted a foundation model strategy, using a pre-trained EndoViT [65] with a frozen encoder while training a lightweight classification head. They classified frames independently and aggregated these into a video-level prediction through majority voting, using adaptive FedSAM [66] for federated optimization to improve generalization. Team Camma employed a metric learning approach featuring a Siamese network with a ResNet-50 backbone [67, 68]. Their model, trained using triplet margin loss, mapped images to embeddings for classification, and they incorporated Switchable Normalization and FedMedian for aggregation to address domain heterogeneity and enhance robustness.

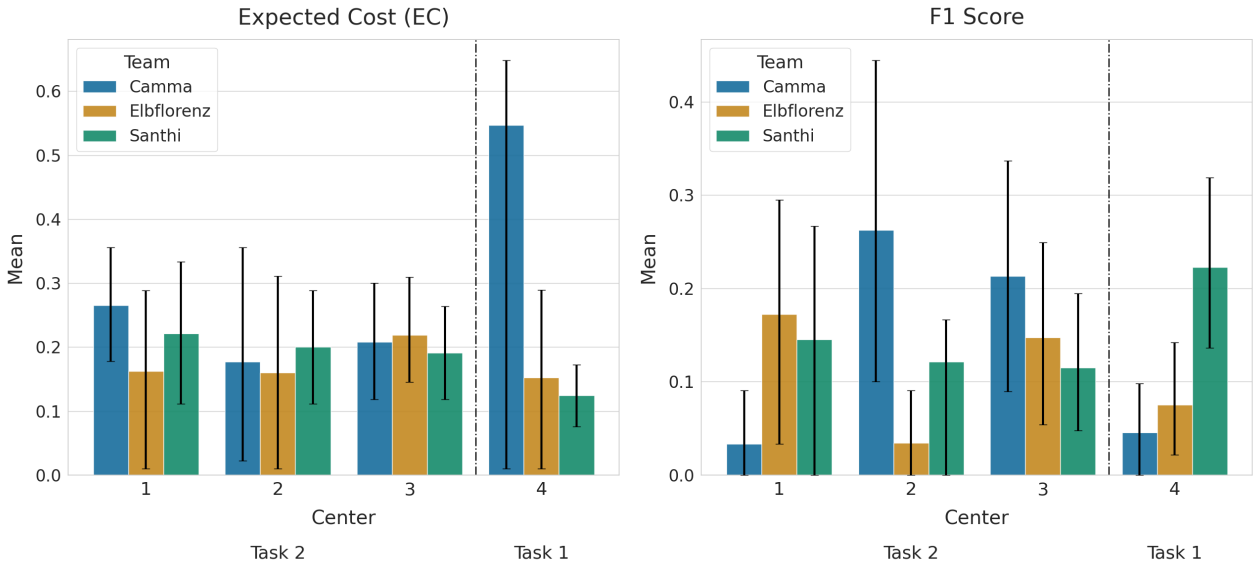
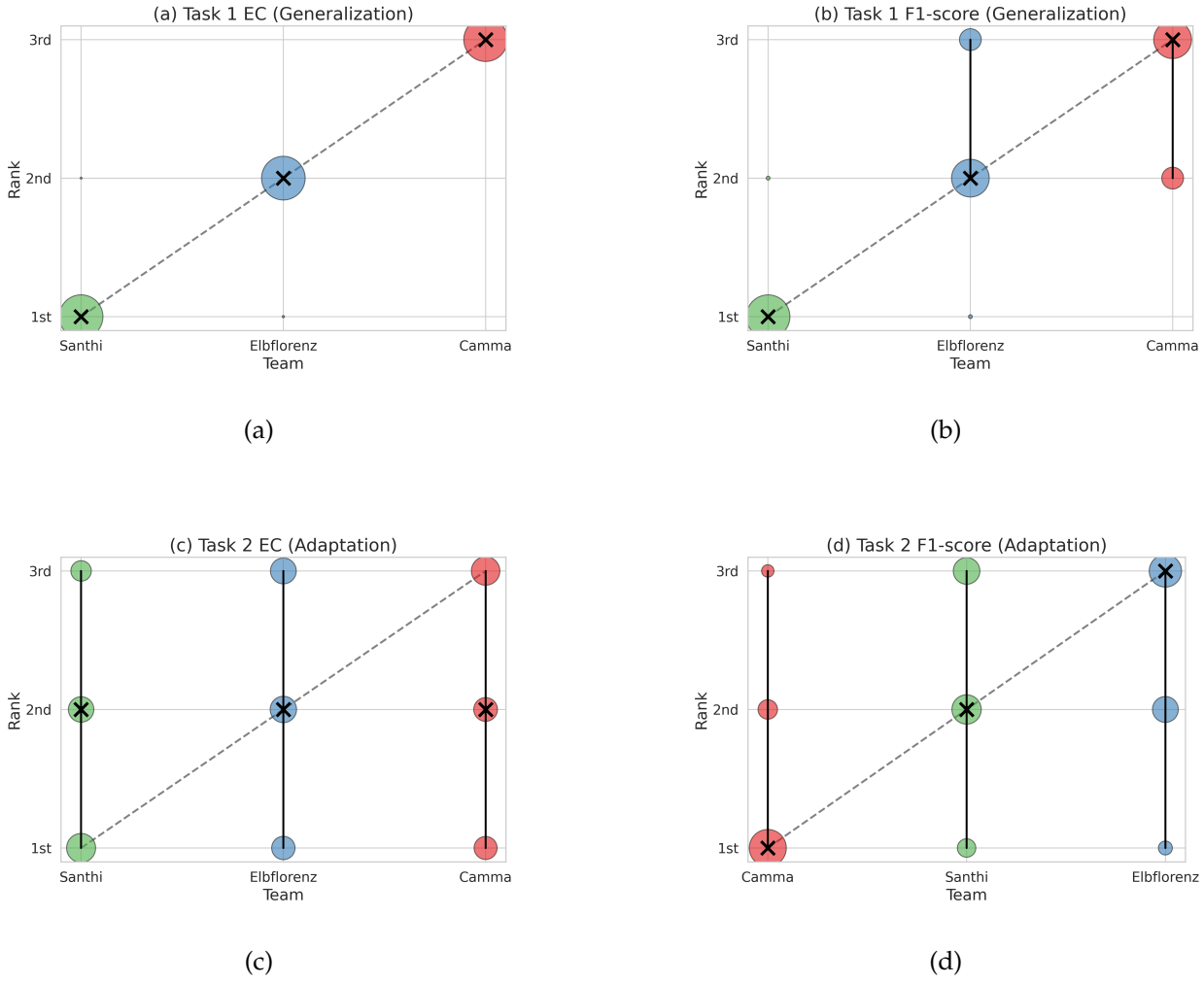



Figure 18: **Bootstrapped Performance Results.** Visualization of the performance results with standard deviation as error bars for all teams and tasks after bootstrapping with 10,000 repetitions. The plot illustrates the variability and stability of the outcomes across different centers.

The final results placed Team Santhi in the first position, as they achieved the top rank in both the generalization and adaptation tasks. Team Elbflorenz and Team Camma tied for the second final rank. In the generalization task, Team Elbflorenz placed second and Team Camma third. In the adaptation task, Team Camma secured the second rank while Team Elbflorenz placed third. However, bootstrapping and statistical analysis revealed that the ranking for task 2 is unstable. The challenge ultimately revealed several critical limitations of FL in this context, including significant difficulties with generalization, managing class imbalance, and effectively tuning hyperparameters within




(a) (b) (c) (d)

Figure 19: **Ranking Stability.** Bootstrapped ranking distributions for each metric and task, based on 10,000 bootstrap iterations. Circle size indicates the percentage of times a team's model achieved a specific rank across samples. Black crosses show median ranks, and black lines denote the 95% bootstrap confidence intervals. Subfigures (a) and (b) correspond to Task 1 (generalization ability) with metrics EC and F1-score, respectively, while (c) and (d) represent Task 2 (adaptation ability) using the same metrics.

decentralized settings. It underscored the importance of advancing future research in federated surgical AI by focusing on improved temporal modeling, context-aware preprocessing, and the design of data-sensitive loss functions. Those results will be soon published in a journal paper. [8]

The FedSurg challenge is a significant component that underscores the project's EU-level relevance. It directly addresses the critical challenge of sharing sensitive medical data across national borders. By employing a federated approach, it provides a viable mechanism for collaborative AI model training among multi-national partners, while ensuring robust data security and strict GDPR compliance. This is possible precisely because the AI model is trained locally at each institution, meaning the sensitive raw patient data never leaves the hospital's secure environment. This achievement is key as it validates a secure and scalable framework for collaboration, effectively safeguarding patient privacy while simultaneously advancing pan-European health research.

#### 4.6.2 FL-EndoViT

This experiment investigates the viability of federated learning (FL) as an alternative to centralized training for developing foundation models in minimally invasive surgery. The study seeks to determine if a foundation model trained via a federated approach can perform comparably to a centrally trained foundation model across various surgical tasks. The work builds upon the established EndoViT model by Batić et al. [65] and incorporates the Sharpness-Aware Minimization (SAM) [66] optimization strategy as adapted by Caldarola et al. for federated environments.

The centralized baseline for this study is EndoViT [65], a model trained in two stages. Initially, it undergoes self-supervised pretraining using a Masked Autoencoder (MAE) on the extensive Endo700k dataset collection. The model develops powerful data representations during its pretraining phase by learning to reconstruct heavily masked image patches, usually 75% of the image. This learning process is significantly enhanced by a heterogeneous data collection, which includes diverse surgical procedures and anatomies. By being exposed to such varied data, the model is compelled to form robust, generalizable representations applicable across different surgical scenarios, rather than simply memorizing features specific to one dataset. Subsequently, the pretrained model is fine-tuned for specific downstream applications, including Surgical Scene Segmentation (SSS), Action Triplet Recognition (ATR), and Surgical Phase Recognition (SPR). Performance is benchmarked using metrics appropriate for each task: IoU for SSS, mAP for ATR, and accuracy for SPR. EndoViT has demonstrated strong performance, often matching or exceeding models pretrained on the ImageNet dataset, particularly in scenarios with limited data.

To adapt this process for a decentralized setting, the study leverages Adaptive Federated Sharpness-Aware Minimization. This approach was designed to address the performance degradation and limited generalization often seen in FL due to heterogeneous data across different clients. The strategy links these issues to the sharpness of the loss landscape. By introducing SAM on the client side, the model training targets flatter regions of the loss landscape to enhance generalization. Concurrently, Stochastic Weight Averaging (SWA) is employed on the server side during the aggregation of client models to promote convergence to these flatter minima.

The proposed Federated EndoViT (FL-EndoViT) framework mirrors the two-stage approach of the original model but adapts it for federated foundation model training. The first stage consists of federated self-supervised pretraining where decentralized centers collaboratively train a shared global model using their local datasets without direct data sharing. Each center has access to one of the dataset of the Endo700k dataset collection [65]. During this phase, each center uses the adaptive FedSAM [69] optimizer to reconstruct masked patches. The resulting client models are then aggregated by a central server using federated averaging (FedAvg). SWA is applied during the final epochs to ensure a smooth and stable final model. In the second stage, the pretrained FL-EndoViT encoder serves as a foundation model for centralized fine-tuning and feature extraction on the SSS, ATR, and SPR tasks.

For implementation, the Endo700k dataset, a composite of nine public surgical datasets, was utilized for pretraining. To simulate a FL scenario and prevent data overlap, task-specific versions of

[8] Pre-print available here: `https://www.arxiv.org/abs/2510.04772`

Table 11: **Pretraining performance comparison:** Pretraining performance comparison across federated optimization methods on the Endo700k dataset. The table reports the distribution of reconstructed patches below different reconstruction loss thresholds out of 256 total patches. Results show that combining MAE pretraining with adaptive FedSAM achieves performance closest to the centralized baseline, highlighting its effectiveness in handling the non-iid and heterogeneous data distributions of the federated setting.

| Method | Threshold 0.3 | Threshold 0.1 | Threshold 0.05 | Threshold 0.01 |
|---|---|---|---|---|
| **Centralized Baseline** | 171 (171) | 126 (120) | 92 (90) | 45 (45) |
| **FedASAM (ours)** | **168 (166)** | **112 (107)** | **82 (80)** | **39 (38)** |
| **FedAvg** | 67 (67) | 25 (25) | 11 (11) | 0 (0) |
| **FedMedian** | 39 (39) | 5 (5) | 0 (0) | 0 (0) |
| **QFedAvg** $(Q = 2)$ | 53 (49) | 8 (7) | 1 (1) | 0 (0) |
| **QFedAvg** $(Q = 0.5)$ | 65 (63) | 20 (16) | 6 (4) | 0 (0) |
| **FedAvgM** | 67 (66) | 24 (24) | 11 (11) | 0 (0) |
| **KRUM** | 65 (65) | 23 (23) | 10 (10) | 0 (0) |
| **FedAdam** | 67 (66) | 24 (24) | 11 (11) | 0 (0) |

the dataset were created, with each FL client having access to only one out of the nine subdatasets of Endo700k. The server operated without any direct data access. The entire process was implemented using the Flower FL Framework [70] with PyTorch, running on NVIDIA V100 GPUs. To ensure comparability, hyperparameters were adopted from the original EndoViT repository.

A series of experiments were conducted to evaluate the framework. An initial ablation study on federated pretraining compared commonly used FL approaches against one enhanced with adaptive FedSAM. The results demonstrated that while common approaches (FedAvg [64], QFedAvg [71], KRUM [72], ...) underperformed, the integration of FedSAM significantly improved reconstruction error and the number of accurately reconstructed patches.

Although this did not fully match the performance of centralized pretraining, it confirmed the benefits of FedSAM and SWA in improving reconstruction. Consequently, the FL-EndoViT model incorporating adaptive FedSAM and SWA was selected for all further evaluations.

Further experiments on the downstream tasks began with an ablation study to determine the best fine-tuning strategy. This study compared the performance of full fine-tuning against freezing the model's backbone and fine-tuning only the task-specific head. The results clearly indicated that freezing the backbone led to suboptimal performance, particularly for the FL-EndoViT model. Full fine-tuning yielded substantial improvements across tasks and was therefore deemed necessary for effective adaptation.

In the final fine-tuning experiments, for Surgical Scene Segmentation, the centralized CEN-EndoViT performed better in low-resolution settings, but FL-EndoViT consistently outperformed it in high-resolution settings, especially with smaller training sets. For Action Triplet Recognition, FL-EndoViT showed superior performance with larger datasets, achieving a notable improvement over the centralized model, while performance was comparable in few-shot settings. Finally, for Surgical Phase Recognition, both the federated and centralized models exhibited similar performance across stages that both excluded and included temporal components, demonstrating the robustness of FL-EndoViT in capturing temporal information.

In conclusion, this study successfully demonstrates that foundation models trained with federated learning can generalize effectively across diverse surgical datasets while preserving data privacy. The integration of adaptive FedSAM during pretraining is shown to be a key factor in enhancing generalization and robustness in a heterogeneous data environment. The findings underscore that for optimal performance, full fine-tuning of both the model backbone and the task-specific head is crucial. The work validates the efficacy of federated foundation models in achieving performance

Table 12: **Impact of Frozen Fine-Tuning in comparison to Fully Fine-Tuning:** This table compares the performance of fully fine-tuned and frozen-backbone models across Surgical Scene Segmentation (SSS), Action Triplet Recognition (ATR), and Surgical Phase Recognition (SPR). The reported values represent the mean metric computed per image or video, followed by the mean and standard deviation calculated across all images. The bold values indicate whether CEN-EndoViT or FL-EndoViT performs better. Statistically significant differences, determined using the Wilcoxon signed-rank test with $\alpha = 0.01$, are marked with an asterisk (*).

| | **Task** | **SSS** | | **ATR** |
|---|---|---|---|---|
| | | **High Res** | **Low Res** | |
| | **Metric** | **IOU** | | **mAP** |
| **Full** | **CEN-EndoViT** | **67.81**% ± **8.03**%* | **67.54**% ± **8.06**%* | 31.33% ± 4.61% |
| | **FL-EndoViT** | 65.98% ± 8.10% | 67.42% ± 8.18% | **40.79**% ± **6.65**%* |
| **Frozen** | **CEN-EndoViT** | **68.57**% ± **8.14**%* | **67.28**% ± **8.94**%* | **31.33**% ± **4.61**%* |
| | **FL-EndoViT** | 67.23% ± 7.37% | 65.22% ± 7.55% | 26.90% ± 5.58% |

| | **Task** | **SPR** | | | |
|---|---|---|---|---|---|
| | | **S1** | **S2** | **S1** | **S2** |
| | **Metric** | **ACC** | | **F1** | |
| **Full** | **CEN-EndoViT** | **81.58**% ± **8.02**% | 88.37% ± 7.16% | **71.47**% ± **8.46**% | 84.48% ± 6.15% |
| | **FL-EndoViT** | 81.46% ± 7.59% | **89.04**% ± **7.03**% | 71.19% ± 8.26% | **85.05**% ± **5.92**% |
| **Frozen** | **CEN-EndoViT** | **72.26**% ± **8.96**%* | **79.88**% ± **10.88**%* | **59.71**% ± **8.70**%* | **75.12**% ± **9.50**%* |
| | **FL-EndoViT** | 65.72% ± 10.72% | 75.26% ± 11.55% | 53.39% ± 8.82% | 72.29% ± 9.46% |

Table 13: **Impact of Few-Shot Fine-Tuning on Surgical Scene Segmentation (SSS):** This table compares the SSS performance of FL-EndoViT and CEN-EndoViT in a few-shot setting across different training set sizes. The reported values represent the mean IoU-Score computed per image over the technical repetitions, followed by the mean and standard deviation caclucation across all images. The bold values indicate weather CEN-EndoViT or FL-EndoViT performs better. Statistically significant differences, determined using the Wilcoxon paired test with $\alpha = 0.01$, are marked with an asterisk (*).

| Fine-Tuned on | Res. | CEN-EndoViT | FL-EndoViT |
|---|---|---|---|
| 1 Video | Low | **42.87**% ± **6.84**%* | 40.50% ± 7.60% |
| | High | 25.68% ± 8.32% | **42.03**% ± **6.50**%* |
| 2 Videos | Low | **56.60**% ± **5.54**%* | 53.32% ± 5.40% |
| | High | 37.12% ± 3.82% | **56.39**% ± **6.39**%* |
| 4 Videos | Low | **61.06**% ± **5.62**%* | 58.98% ± 6.25% |
| | High | 45.51% ± 5.61% | **60.34**% ± **6.58**%* |

Table 14: **Impact of Few-Shot Fine-Tuning on Action Triplet Recognition (ATR):** This table compares the ATR performance of FL-EndoViT and CEN-EndoViT in a few-shot setting across different training set sizes. The reported values represent the mean average precision score computed per image over the technical repetitions, followed by the mean and standard deviation caclucation across all images. The bold values indicate weather CEN-EndoViT or FL-EndoViT performs better. Statistically significant differences, determined using the Wilcoxon paired test with $\alpha = 0.01$, are marked with an asterisk (*).

| Fine-Tuned on | CEN-EndoViT | FL-EndoViT |
|---|---|---|
| 2 Videos | **22.61**% ± **3.04**% | 21.76% ± 4.30% |
| 4 Videos | 26.00% ± 4.18% | **27.24**% ± **4.75**% |
| 8 Videos | **33.16**% ± **6.41**% | 32.53% ± 5.71% |

Table 15: **Impact of Few-Shot Fine-Tuning on Surgical Phase Recognition (SPR):** This table compares the SPR performance of FL-EndoViT and CEN-EndoViT in a few-shot setting across different training set sizes. The reported values represent the accuracy and F1-Score computed per video over the technical repetitions, followed by the mean and standard deviation calcucation across all videos. The bold values indicate weather CEN-EndoViT or FL-EndoViT performs better. Statistically significant differences, determined using the Wilcoxon paired test with $\alpha = 0.01$, are marked with an asterisk.

| Metric | | Accuracy | | F1-Score | |
|---|---|---|---|---|---|
| Fine-Tuned on | TeCNO Stage | CEN-EndoViT | FL-EndoViT | CEN-EndoViT | FL-EndoViT |
| 2 Videos | S1 | **63.83**% ± **8.94**%* | 61.30% ± 8.51% | **48.95**% ± **8.26**%* | 45.70% ± 6.96% |
| | S2 | **78.13**% ± **8.94**% | 76.35% ± 10.53% | 75.15% ± 6.86% | **75.22**% ± **8.13**% |
| 4 Videos | S1 | **66.79**% ± **9.92**% | 66.23% ± 9.15% | 52.50% ± 10.00% | **53.22**% ± **8.94**% |
| | S2 | 79.75% ± 9.55% | **80.07** ± **8.33**% | 77.67% ± 7.50% | **77.89**% ± **8.04**% |
| 8 Videos | S1 | **74.04**% ± **8.65**%* | 71.96% ± 10.32% | 58.76% ± 9.90% | **60.62**% ± **10.02**%* |
| | S2 | 84.34% ± 7.94% | **84.61** ± **8.33**% | 80.73% ± 6.86% | **81.55**% ± **6.46**% |

Table 16: **Fine-Tuning Performance on Classification of Unseen Data:** This table compares the performance of CEN-EndoViT and FL-EndoViT after fine-tuning on three representative downstream tasks of the GynSurg dataset: action recognition, bleeding detection, and smoke detection. Reported metrics are accuracy and F1-score (mean ± standard deviation) after bootstrapping with 10,000 repetitions. The bold values indicate weather CEN-EndoViT or FL-EndoViT performs better. Statistically significant differences, determined using the Wilcoxon paired test with $\alpha = 0.01$, are marked with an asterisk.

| **Fine-Tuned on** | **CEN-EndoViT** | | **FL-EndoViT** | |
|---|---|---|---|---|
| | **Accuracy** | **F1-Score** | **Accuracy** | **F1-Score** |
| **Action** | 49.74% ± 3.25% | 38.64% ± 3.42% | **65.88**% ± **3.01**%* | **57.93**% ± **3.68**%* |
| **Bleeding** | 78.00% ± 3.52% | 76.74% ± 3.84% | **87.24**% ± **2.97**%* | **86.72**% ± **3.19**%* |
| **Smoke** | 82.29% ± 1.68% | 82.28% ± 1.67% | **87.38**% ± **1.44**%* | **87.36**% ± **1.45**%* |

that is comparable, and in some cases superior, to traditional centralized training methods [9].

### 4.7 High Levels of Data Security and Confidential Computing Validated Using TEEs and Federated Learning in Adversarial Security Experiments

Federated Learning (FL) enables collaborative model training while preserving data confidentiality and integrity. The integration of the SCONE secure container environment leverages Trusted Execution Environments (TEEs) to protect both computation and data, even in adversarial settings. Recent SCONE releases introduce advanced features such as dynamic attestation, fine-grained policy enforcement, and confidential orchestration, further strengthening the security posture of FL deployments.

**Security Validation via Memory Inspection**

To validate SCONE's security guarantees, we employ a memory inspection methodology. In Linux, process memory can be accessed via the `/proc` filesystem, specifically through the `mem` and `maps` files associated with a process ID (PID). For example, if the FL process has PID `1310177`, an adversary with privileged access can attempt to read `/proc/1310177/mem` and `/proc/1310177/maps`, dumping their contents for analysis.

The security assertion proceeds in two phases:

1. **Memory Dumping:** The adversary dumps the process memory to a file.
2. **Secret Search:** Knowing a secret value (e.g., the string "SURGERY") that should be present in memory during non-confidential execution, the adversary searches the dump using tools like `grep`. In a properly secured (confidential) execution, this secret should not be found in the memory dump, demonstrating that sensitive data remains protected even against privileged attackers.

**Experiment 1: Federated Learning within SGX Enclaves Using SCONE and CAS**

**Objectives:**

- Assess how confidential federated learning with TEEs mitigates key security challenges in FL, including threats from privileged attackers, colluding malicious clients, and adversarial manipulation of local updates.
- Measure the impact of SCONE's network shield (configured via the Data Broker's Configuration and Attestation Service, CAS) on FL execution time, compared to both a standard (non-SGX) environment and a basic SGX-enabled setup.
- Analyze the effect of SGX-based optimizations on the startup latency of FL application functions.
- Evaluate the benefits of new SCONE features such as dynamic attestation and confidential orchestration.

**Experimental Setup**

- **Hardware:** Intel Xeon® CPU Silver 4314 @ 2.40 GHz, 128 GiB RAM, 500 GB SATA SSD.
- **Software:** Ubuntu 20.04, Docker 20.10.19, SCONE v6.0+ with enhanced attestation and policy features.
- **Environment:** All experiments conducted within Docker containers orchestrated via Kubernetes with SCONE Confidential Orchestration.

| Scenario | Description |
|---|---|
| Vanilla (Intel/no SGX) | FL application running on Intel CPUs without SGX support. |
| SCONE (Intel SGX/hardware mode) | FL application running on Intel CPUs with SGX, benefiting from hardware-backed security guarantees. |
| SCONE (SGX/hardware mode) + CAS + New Features | FL application running with SCONE and CAS, enabling advanced security features:<br>**Network Shielding:** Protects network connections between FL clients and servers.<br>**File Shielding:** Encrypts input training data using SCONE's enhanced file system shield, now supporting dynamic key rotation and granular access policies.<br>**Dynamic Attestation:** Continuous verification of enclave integrity and policy compliance during runtime.<br>**Confidential Orchestration:** Secure deployment and scaling of FL workloads using Kubernetes, with all orchestration metadata and secrets protected within enclaves. |

Table 17: Evaluated Scenarios

**Evaluated Scenarios**

**Results**

As shown in Figure 20, the primary overhead introduced by SCONE in SGX mode, compared to the vanilla environment, is due to enclave startup. This startup time is largely fixed for a given enclave size. For more complex or longer-running applications, this overhead becomes a smaller proportion of the total execution time.

**Security Features:**

- **Network Shield:** Configured via the Data Broker's CAS, this feature ensures that all network traffic between FL clients and servers is encrypted and authenticated.
- **File System Shield:** SCONE encrypts all input training data at rest, decrypting it only within the secure enclave for processing. The latest version supports dynamic key rotation and fine-grained access control, reducing the risk of key compromise.
- **Dynamic Attestation:** SCONE now supports continuous attestation, ensuring that enclaves remain in a trusted state throughout their lifecycle. Any deviation from expected measurements or policies triggers automatic revocation of access to secrets and data.
- **Confidential Orchestration:** With Kubernetes integration, SCONE ensures that deployment metadata, secrets, and configuration files are only accessible within enclaves, preventing leakage even if the orchestrator is compromised.

**Function Startup Optimization**

To further analyze performance, we measured the startup time of a no-op enclave as a function of the allocated heap size (`SCONE_HEAP`). Results are summarized below:

These results indicate that enclave startup time decreases as heap size increases, but remains a significant factor in total application runtime, especially for large enclaves.

**Conclusion**

The proposed solution ensures that both input training data and code (e.g., Python scripts) are encrypted and that all training computations—local and global—are executed within TEE-protected

[9]This paper is already available as preprint here: `https://arxiv.org/abs/2504.16612`

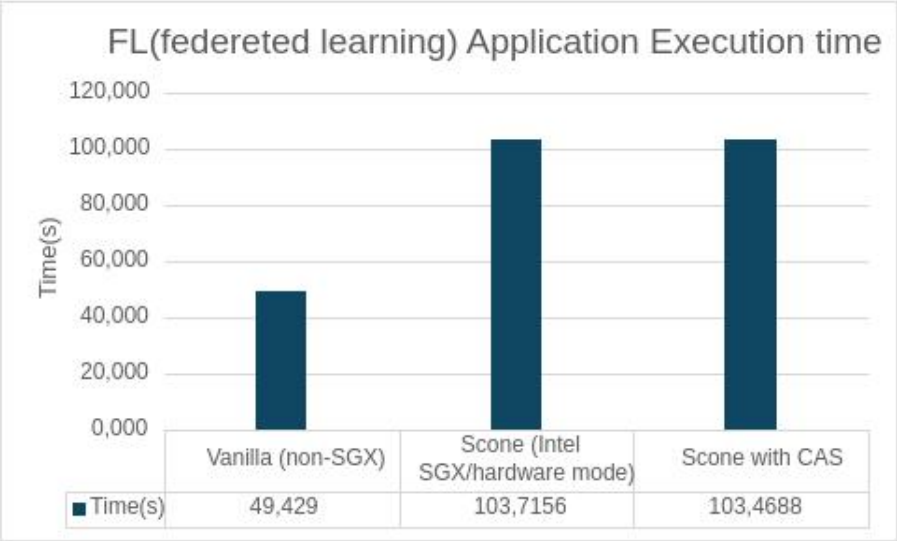



Figure 20: FL Execution Time

| Heap Size | Startup Time (s) |
|---|---|
| 6 GB | 220.69 |
| 8 GB | 210.15 |
| 10 GB | 209.13 |
| 12 GB | 193.66 |

Table 18: Startup time of a no-op enclave depending on the dedicated heap size for SGXv1

enclaves. All gradient updates are transmitted over Transport Layer Security (TLS) connections between client and server enclaves, ensuring end-to-end confidentiality and integrity.

The Data Broker, integrated with the CAS component, leverages remote attestation to guarantee that only authorized code and data are used in training. With the latest SCONE features, dynamic attestation and confidential orchestration further ensure that computations are performed with the correct, unmodified code and data, and that secrets are never exposed outside trusted enclaves.

Preliminary evaluations demonstrate that this approach maintains the confidentiality and integrity of federated learning computations without sacrificing model utility or accuracy. Attackers, even with privileged access, are unable to compromise the integrity or confidentiality of the training data, code, or models, validating the effectiveness of SCONE and TEE-based confidential computing in adversarial settings.

**Key Takeaways:**

- SCONE and TEEs provide robust protection for federated learning, even against privileged attackers.
- Memory inspection experiments confirm that sensitive data remains confidential within enclaves.
- Performance overhead is primarily due to enclave startup, which can be mitigated for longer-running workloads.
- The integration of CAS, dynamic attestation, and confidential orchestration ensures the integrity and authenticity of the entire FL workflow.

- Enhanced file system shield and policy enforcement features in SCONE further reduce the attack surface and support compliance requirements.

### 4.8 Use-case repositories

- https://github.com/KirchnerMax/FL-EndoViT
- https://gitlab.com/nct_tso_public/challenges/miccai2024/FedSurg24
- https://gitlab.com/nct_tso_public/challenges/miccai2024/snippet

### 4.9 Discussion and final remarks

We demonstrated the feasibility of federated learning for training foundation models in surgical vision, showcasing its potential to overcome data-sharing limitations while achieving comparable or superior performance to centralized approaches. The FedSurg challenge further solidified the importance of FL in addressing real-world surgical AI challenges. We also highlighted the advantages of utilizing Pravega and GStreamer for efficient handling of high-volume, real-time surgical video data, demonstrating significant improvements in data loading speed compared to traditional methods. These advancements contribute significantly to enabling robust and privacy-preserving AI solutions in computer-assisted surgery.

### 4.10 Future work

The insights gained from this project, particularly regarding federated learning and efficient video stream handling, will serve as a crucial foundation for subsequent initiatives such as the Surgical AI Hub Germany. This collaborative venture aims to further advance the application of AI in surgery. A key area for future work involves deploying and testing streaming technologies like Pravega and GStreamer directly within the operating room, providing a platform for real-world validation and surgeon assistance. Following this, the implementation of federated learning is planned within the Surgical AI Hub Germany to translate these research advancements directly into clinical practice. Furthermore, future work will address pressing clinical needs identified by surgeons, including the automated identification and analysis of adverse events and the autonomous generation of surgical reports. The streaming and AI technologies developed will be leveraged to address these critical requirements, aiming to provide valuable clinical insights and streamline essential documentation processes.

# 5 Use Case: Transcriptomics

## 5.1 Short Description

The use case is organized into two key experiments, each representing a separate scenario with different challenges and datasets. The experiments are listed below:

- Transcriptomics Atlas (processing of RNA-seq data)
- Federated Learning for Genomics Data Analysis

Transcriptomic analyses are most often performed in a comparative framework, examining different health states, diseases, or stimuli relative to baseline conditions. However, acquiring data under standardized conditions substantially increases experimental costs [73]. In addition, the absence of such data during the planning phase complicates the optimization of experimental design. The solution to those challenges is the **Transcriptomics Atlas**, in which data from a representative set of human tissues were uniformly processed. This resource can be of use in a wide range of scientific applications, including pharmacogenomics and biomarker discovery.

The Transcriptomics Atlas pipeline steps are described below:

1. Downloading SRA file using *prefetch* tool.
2. Converting into FASTQ files using *fasterq-dump* tool.
3. Alignment and quantification of reads using *Salmon* [74] or *STAR* [75] [76].
4. Count normalization using *DESeq2* [77].

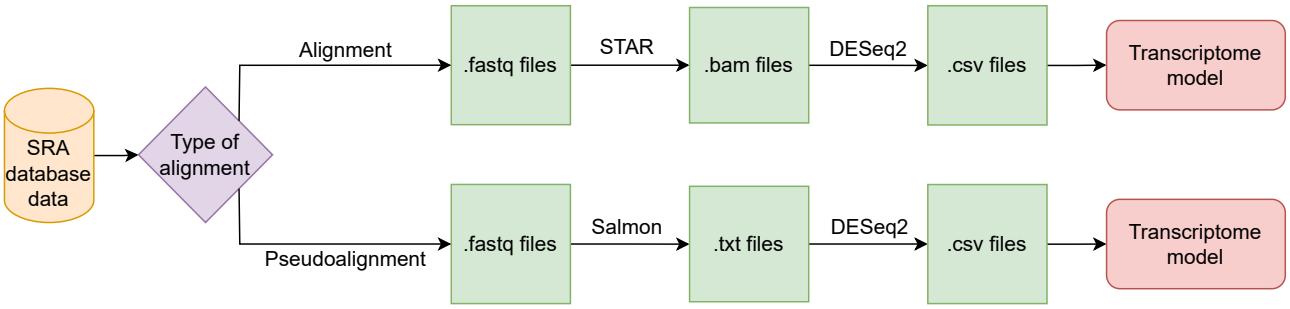



Figure 21: Transcriptomics Atlas Pipeline.

Alignment of RNA-sequences is the most expensive step in terms of compute resources and time. Processing hundreds of terabytes of such data is challenging and requires a cost-efficient approach and thoughtful optimizations. The cloud architecture designed for this task is presented in Fig. 22. The pipeline itself is deployed on EC2 worker-nodes within AutoScalingGroup.

The second key experiment was initially designed to apply a federated learning (FL) framework to Human Genome Variation Analysis. This domain was selected as a canonical use case for FL, as human genomic data is exceptionally sensitive and strictly regulated, forcing it to remain siloed within different research institutions. This presents a perfect challenge for an FL model, which can train collaboratively without moving or exposing the raw data. However, procuring a suitable, distributed, and well-annotated human genomic variation dataset for this project proved to be a significant logistical challenge. We therefore made a decision pivot this experiment to microbiome data, for which we had a readily available model and dataset. Microbiomics domain preserves the original experiment's core technical challenge. Microbiome data (i.e., metagenomic data) faces privacy and

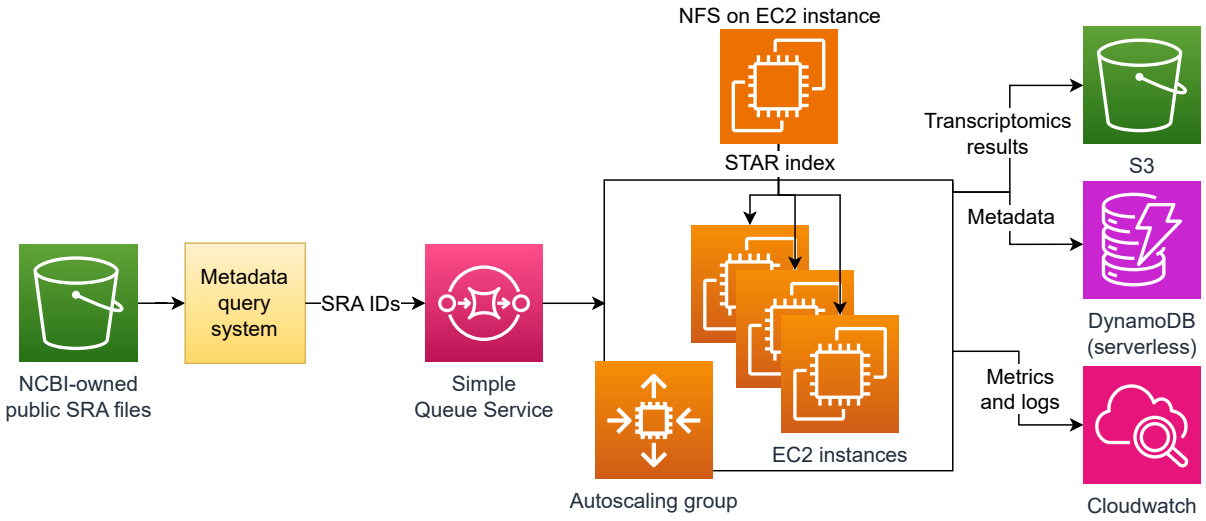



Figure 22: Cloud architecture for Transcriptomics Atlas Pipeline.

logistical hurdles that are highly similar to those of human genomic data. A person's microbiome is a unique fingerprint that can be used for identification and reveals sensitive health information about their diet and lifestyle. Therefore, this data is also siloed within institutions and subject to stringent data-sharing restrictions, making it an ideal use case for demonstrating the viability of federated learning for privacy-preserving analysis.

## 5.2 Data Spaces

### 5.2.1 Transcriptomic Atlas

In this part of the Use Case, we focus on data available from the NCBI Sequence Reads Archive repository [78] and European Nucleotide Archive (ENA). The available dataset size exceeds 30 petabytes of sequence data [79] [80]. However, we select RNA-sequence data generated by human samples sequencing only, based on tissue/cell, lack of disease, and appropriate technical parameters of sequencing. The files themselves are hosted on AWS (us-east-1), with computations performed in the same (or close-by) region for proximity and efficiency. The data for the Transcriptomics Atlas consists of sequences of 20 human tissues. In total, our dataset consists of 17TB of SRA data (compressed) which corresponds to 130TB of FASTQ data (uncompressed). Extending the dataset is possible with additional tissues in order to create a broader Transcriptomics Atlas.

### 5.2.2 Federated Learning experiments

For the federated learning experiment, the model integrated with our FL framework is based on DeepMicro [81]. This model is compatible with several publicly available human gut metagenomic datasets, such as those aggregated in the MetAML repository [82]. These datasets consist of whole-genome shotgun (WGS) metagenomic samples from six different disease cohorts: inflammatory bowel disease (IBD), type 2 diabetes in European women (EW-T2D), type 2 diabetes in Chinese (C-T2D), obesity (Obesity), liver cirrhosis (Cirrhosis), and colorectal cancer (Colorectal). All samples were derived from Illumina paired-end sequencing technology, and each cohort contains a mix of healthy controls and patient samples from different studies. In total, these six cohorts comprise 1,156 human gut metagenomic samples.

### 5.3 Targeted KPIs

This work is related to the following project KPIs:

- **KPI-1 - Significant performance improvements**: We improved the Transcriptomics Atlas pipeline with cloud-related and application-specific optimizations that resulted in two-order of magnitude cheaper computations (full analysis in Tab. 19).
- **KPI-3 - Demonstrated resource auto-scaling for batch and stream data processing**: We utilized services such as AWS EC2 AutoScalingGroup, ECS in Fargate mode, AWS Lambda which allowed to easily scale resources for the current need.

### 5.4 Challenges addressed

#### 5.4.1 Transcriptomic Atlas

The main challenge in this part of the Use Case is to design cost-efficient batch processing of RNA-sequences for resource-intensive pipeline. Below, we outline the challenges we encountered.

- **Resource-intensive application:** STAR aligner requires indexed human genome to be loaded into worker node's memory in order to perform alignment on a FASTQ file. By using the newest release of the genome we were able to reduce the requirement from 80GB to 35GB therefore save resources, reduce costs and improve significantly processing time.
- **Large-scale dataset of RNA-sequences:** For the Transcriptomics Atlas use case we need to process 130TB of FASTQ data. With the optimizations (Tab. 19) implemented this was very efficient and that allows possible extension of the Atlas with additional tissues.
- **Index distribution to worker nodes:** The STAR index which is a 30GB file has to be distributed quickly and efficiently to worker-nodes in order to prevent under-utilization of resources.

#### 5.4.2 Federated Learning experiments

The primary challenge addressed by integrating a federated learning (FL) framework is the ability to perform large-scale machine learning analysis on sensitive, distributed datasets while preserving patient privacy.

- **Enhanced Privacy and Data Governance:** A person's microbiome is a unique "fingerprint" that contains highly sensitive Protected Health Information (PHI). It can be used to identify an individual and reveals sensitive data about their health, diet, and lifestyle. Moving this raw data to a central server for analysis creates a significant privacy risk and is often prohibited by data governance regulations like GDPR and HIPAA. The FL framework directly addresses this by training the model locally on the "siloed" data at each institution. Only the anonymous model updates (e.g., weights) are shared, not the raw data itself, thus enforcing privacy by design.
- **Overcoming Data Silos:** Microbiome datasets, particularly those for specific diseases, are often small and held by different hospitals or research centers worldwide. It is logistically and ethically difficult, if not impossible, to pool this data. Federated Learning provides a robust framework to collaboratively train a powerful, global model using all the available distributed data without any institution having to share its private dataset.

## 5.5 Data connectors developed

### 5.5.1 Indexed genome connector

Each worker node that performs alignment with STAR must have access to the indexed human genome. From multiple possible solutions, we decided on an NFS-based approach due to better performance than alternatives (e.g. using object storage or embedding the index into the virtual machine image). Earlier versions of the pipeline used a custom NFS server deployed among the worker nodes. The improved version uses the Elastic File System service, which reduced/removed data transfer and operational costs compared to the previous approach.

Furthermore, extensive research has been carried out to evaluate the feasibility and performance impact of a novel index access strategy. The central research question is as follows: given that certain portions of the index are frequently accessed while others are rarely utilized, is it possible to distribute only the core of the index to worker nodes while retrieving the remaining parts remotely? Such an approach could reduce the memory requirements of the nodes; however, remote access is expected to introduce performance overhead. Based on the Fig. 23 we can determine that while the idea is valid since 20% of index consists of roughly 80% of total index accesses. Early research results on this topic were published in [83]. However, more research is needed to decide if performance degradation due to increased latency is worth the memory reduction. Initial results suggest that this idea is more suitable for HPC deployments with fast interconnects.

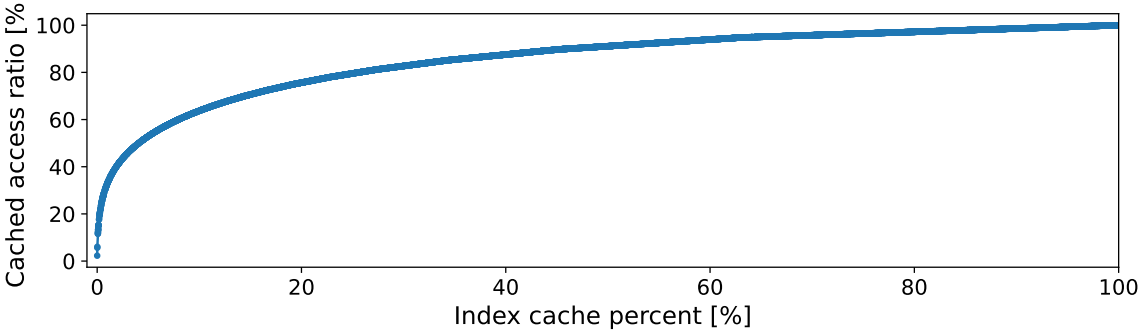



Figure 23: Index access pattern. Relation between size of the local index and local/remote access ratio.

### 5.5.2 Dataplug - SRA format

Since SRA is the default input format for pipelines, while the first step is usually conversion to FASTQ format, we developed an SRA-to-FASTQ extension for the Dataplug library. It allows for seamless partitioning and distribution of chunks of FASTQ file, while the data on the storage remains in SRA format. The main challenge lies in the fact that SRA is a compressed format, requiring a decompression tool to generate FASTQ files. The default method is downloading whole SRA file and converting it to FASTQ locally. To overcome this our approach establishes a mapping between FASTQ lines and the corresponding byte ranges in the original SRA file. This mapping is constructed by monitoring which byte ranges are accessed during decompression, allowing us to link the required bytes to the resulting FASTQ lines without needing detailed insight into the internal mechanics of the decompression tool.

To implement this we slightly modified the NCBI VDB library, which can be used for accessing SRA file. Since the library uses a layered architecture that interfaces with Linux system calls at the lowest level, these calls can be overridden to track which byte ranges are accessed during decompression. This enables precise mapping between compressed data and the logical units of the decompressed format. The method is termed system-level data slicing because it leverages syscalls monitoring to establish the link needed for efficient random access and logical partitioning of SRA data when converting to FASTQ.

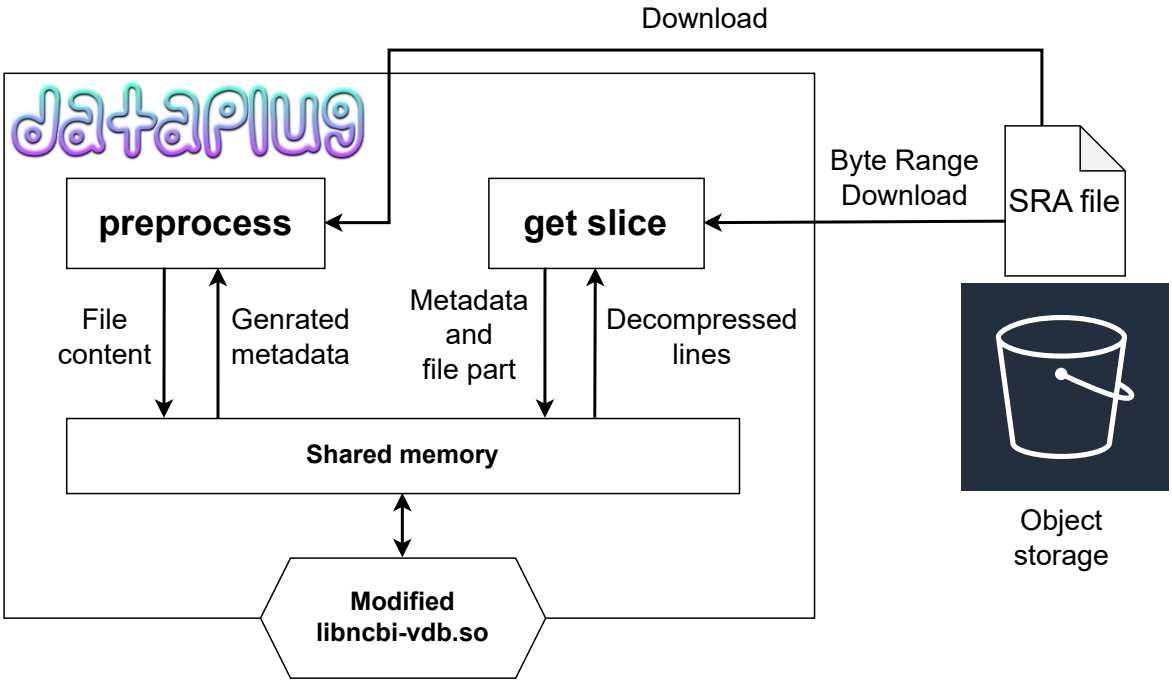



Figure 24: Schema of Dataplug SRA extension interaction with libncbi-vdb.so.

### 5.5.3 VFS and MPI Connectors

We have created 2 solutions for distributed memory access and compared them with the baseline using local memory. Our first solution uses the LD_PRELOAD mechanism to replace local memory access with the distributed one, keeping built-in memory allocation functions. We cannot use kernel customizations due to the lack of root-level access, so we have developed a custom library based on mmap(). It tracks memory allocations and is backed by custom-developed virtual file-system (VFS) stored on on Lustre. We have also developed a solution based on MPI which features one-way communication based purely on RDMA mechanism. We hypothesize that an MPI based solution would provide better performance than VFS. However it cannot be used with proprietary software due to the need of application code instrumentation. Those architectures are shown in Fig. 25 which also compares them with local non-distributed memory access (A) as baseline for the evaluation of our solutions.

The solutions have been tested on two HPC Clusters offered by the PLGrid Infrastructure - Ares and Helios. Each of those clusters offers different type of interconnect - Infiniband and Slingshot. We are presenting results in Fig. 26. As expected the local access is faster than any remote mechanism. The VFS is prone to severe performance degradation as seen in the plot. We expect that this can be improved with faster shared storage or interconnect. On the other hand we have also found out that MPI-based memory access is comparable to local access based on memcpy() which makes it best suited for our goal.

## 5.6 How AI is enabled in the use-case

While the transcriptomics atlas does not heavily rely on AI methods and tools, there are places where AI has been used. In the federated learning experiment, we use a deep learning model (DeepMicro) which is based on deep learning to train an autoencoder to build a deep representation for disease prediction based on human microbiome data. We extend this model by using federated leaning approach, thus helping preserve the data privacy. Our developed Flower-Lithops simulation backend is a contribution towards enabling serverless experiments with federated learning frameworks in a

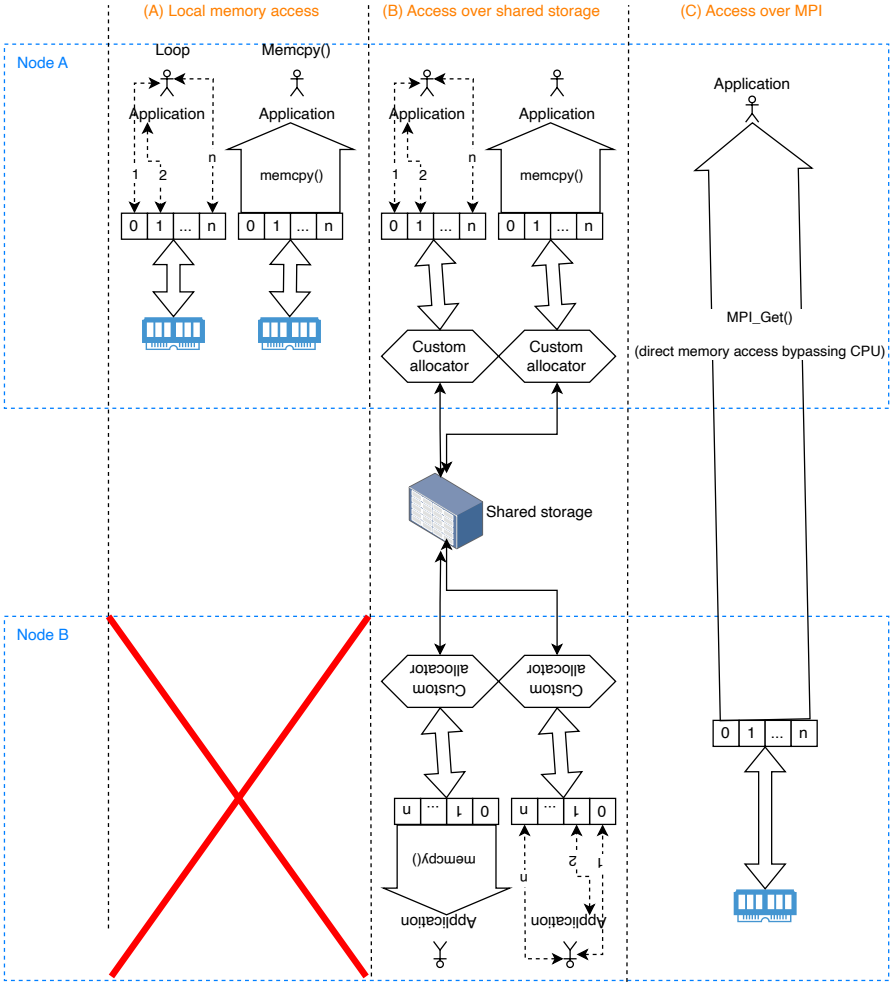



Figure 25: Proposed architectures for memory access - a baseline and two distributed.

more automated and flexible way thanks to Lithops capabilities. Moreover, we have started initial experiments with using LLM-assisted execution planning and optimizing of bioinformatics batch processing, in order to see how the architectures and optimizations we implemented in our use case can be automated with the help of Large Language Models.

### 5.7 Integration with NEARDATA architecture components

As transcriptomics sequences in FASTQ files are independent, a natural step is to partition them into chunks and process them separately. This makes the problem highly parallelizable, opening the door to utilizing the massive parallelism of serverless platforms. We developed two solutions to explore this approach. The first uses Lithops together with Dataplug to process FASTQ.gz files in the AWS cloud. In this setup, Lithops is configured with AWS Lambda as the compute backend. Input data must be stored in S3 in FASTQ.gz format and preprocessed with Dataplug, after which Lithops distributes the computation, collects the outputs, and assembles the final result.

The second solution builds on the first but adapts it to high-performance computing environments. It utilizes the LithopsHPC extension, which we adapted to the Ares supercomputer at ACC Cyfronet. In this version, DataplugSRA is used to eliminate the need to prepare FASTQ files in S3, since SRA files are already publicly accessible on AWS. This adaptation allows efficient integration of Dataplug with both cloud-native serverless workflows and HPC infrastructures.

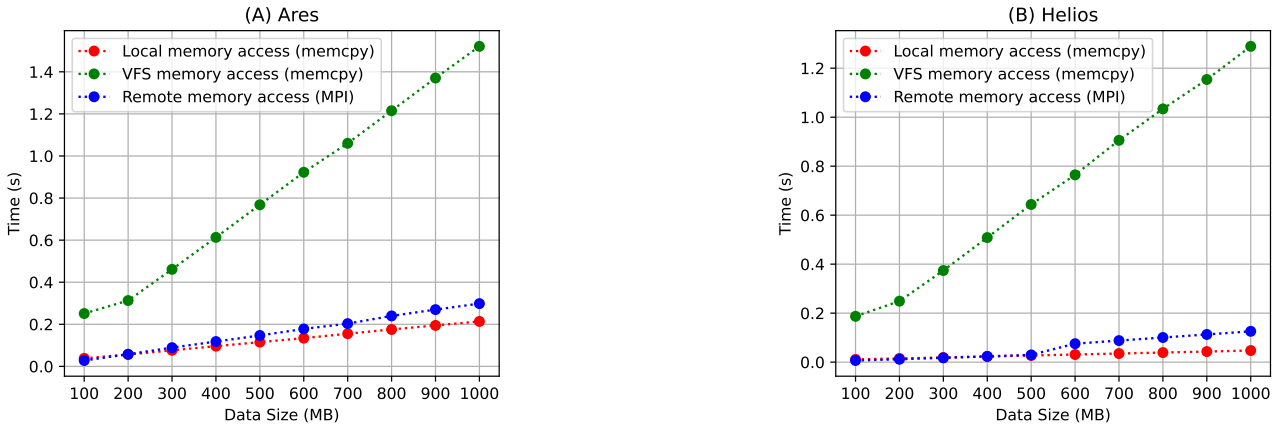



Figure 26: Local memory access vs VFS vs MPI RDMA on: (A) Ares (Infiniband), (B) Helios (Slingshot).

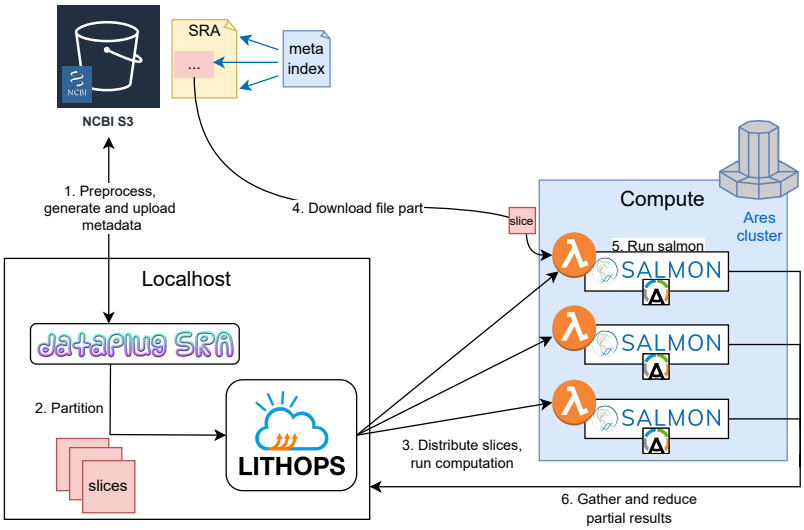



Figure 27: Overview of serverless Salmon pipeline with LithopsHPC.

### 5.7.1 Flower-Lithops Serverless Integration

We integrated the Lithops serverless execution framework with the Flower federated learning framework. This integration unlocks the ability to conduct large-scale federated learning simulations without the need for a standard, persistent compute cluster, such as Flower's default Ray-based backend. Instead of provisioning and managing stateful infrastructure, this integration allows researchers to execute thousands of federated learning clients as ephemeral, on-demand serverless functions. This serverless approach fundamentally changes the accessibility and scale of FL experimentation. It can remove the significant operational overhead of infrastructure management, allowing researchers to focus on model development rather than cluster deployment. This unlocks a new level of elastic scalability, making it feasible to simulate tens of thousands of clients automatically—a scale that would be cost-prohibitive or logistically impractical for many to provision using traditional, persistent clusters. Furthermore, it introduces a highly efficient pay-per-use cost model, which eliminates the expense of idle cluster resources, a common issue with the baseline approach. The development branch for this feature is available at `https://github.com/janprz/flower/tree/add-lithops-backend-support`.

### 5.8 Results and KPIs achieved

The optimization of Transcriptomics Atlas has been a success. In total, the processing time and cost have been reduced by two orders of magnitude. The summary of all optimizations is presented in Tab. 19. Moreover, we processed 130TB of FASTQ data representing RNA-seq data from 20 different human tissues.

Table 19: Summary of architectural and optimization benefits.

| Optimization | Benefit |
|---|---|
| Cloud-native architecture | Scalability, flexibility, cost-efficiency, lower operational overhead |
| STAR index generated on newer release of Ensembl GRCh38 reference gene set | ∼12 times faster STAR alignment time<br>∼8.4 times reduced cloud costs<br>Reduced worker-node memory requirement (80GB -> 32GB)<br>Cheaper and faster index distribution to worker-nodes. |
| STAR Early Stopping | Significant (∼20%) STAR pipeline throughput increase. |
| Index on NFS / EFS | Fast and scalable index distribution solution in the cloud.<br>Reduced data transfer costs (∼10% of the total costs). |
| Scalability analysis of aligners | Improved efficiency of computations.<br>Improved throughput for both Salmon and STAR pipelines |
| Instance type analysis | Improved cost-efficiency of the solution using<br>the most optimal instance type (x86 architecture).<br>R7a.2xlarge is $\geq$ 10% cost-efficient than alternatives. |
| Spot instances usage | Reduction of ∼50% on compute costs.<br>Robust solution to handle interruptions. |
| EBS optimizations (right-sizing) | Improved cost-efficiency of EBS volumes |

#### 5.8.1 KPI-1: Significant performance improvements (data throughput, data transfer reduction).

**Early stopping** is an optimization we introduced in order to save resources by terminating the processing of low-quality files (which cannot be estimated accurately before alignment) or invalid sequences. STAR supports this by measuring intermediate mapping rate, which allows us to identify such sequences. As shown in [84], using live metrics can boost alignment throughput by up to 19.5%. The examples when such terminations occur are presented in Fig. 29.

For the Transcriptomics Atlas project, the mapping rate threshold is set at 30%. However, this is highly dependent on the use case. Pipelines that utilize STAR in a similar scenario and require sequences of the highest quality will greatly benefit from this feature. The usefulness of this approach shows that other aligners should also support monitoring of intermediate mapping rate which currently is not the case. In Fig. 28, we provide an extended analysis based on the experiment reported in [84]. Setting the threshold at 80% results in a 60% reduction in total computation time, indicating that this feature becomes particularly critical when a high mapping rate is desired.

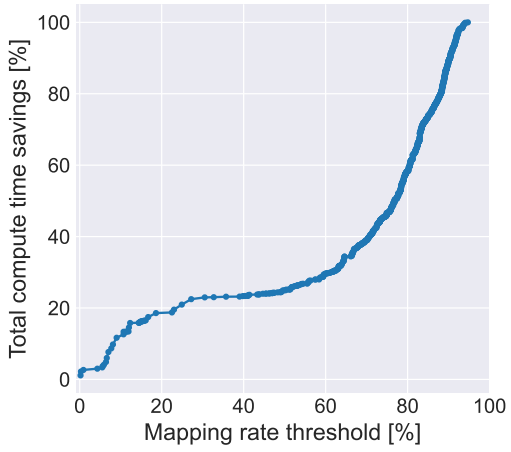



Figure 28: Impact of early stopping threshold on total alignment time.

**Execution time reduction due to usage of newer release of human genome.** STAR alignment step requires an index to be loaded into memory. Such index is generated on the human genome available on e.g. Ensembl [85]. Index generation is a one-time task and then it can be distributed to workers in

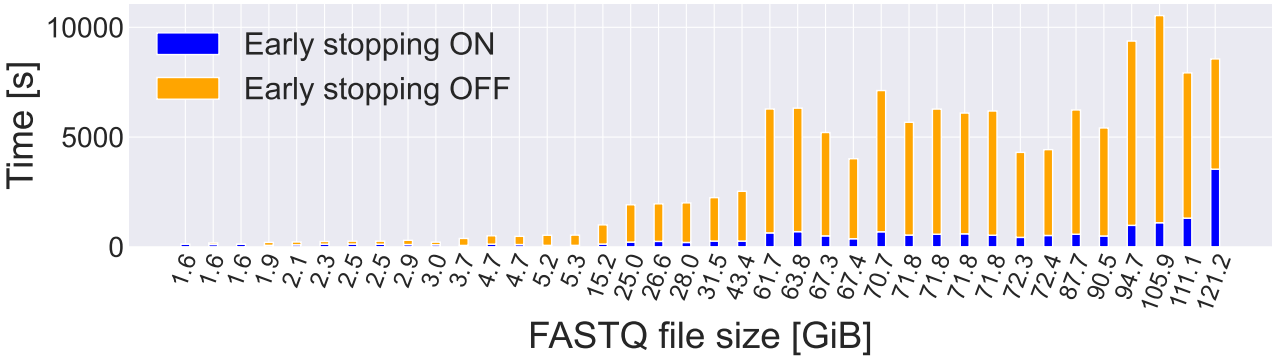



Figure 29: Time savings due to early stopping feature.

the initialization phase. By using a newer release (with a smaller number of patches) of the genome, we generate a much smaller STAR index: 28.5GiB instead of 90GiB. This allows to reduce data transfer cost and time during the initialization phase of each worker. Since each worker loads this index to RAM, we can use smaller instances to run the pipeline. On top of the memory optimizations, we also get significant performance improvement by using a smaller index - over 12 times faster on average (weighted by FASTQ size). We performed an A/B test for 49 *SRA* files and ran the pipeline with different index releases. The execution times are presented on Fig. 30. This is applicable almost exclusively to the STAR aligner path of the pipeline.

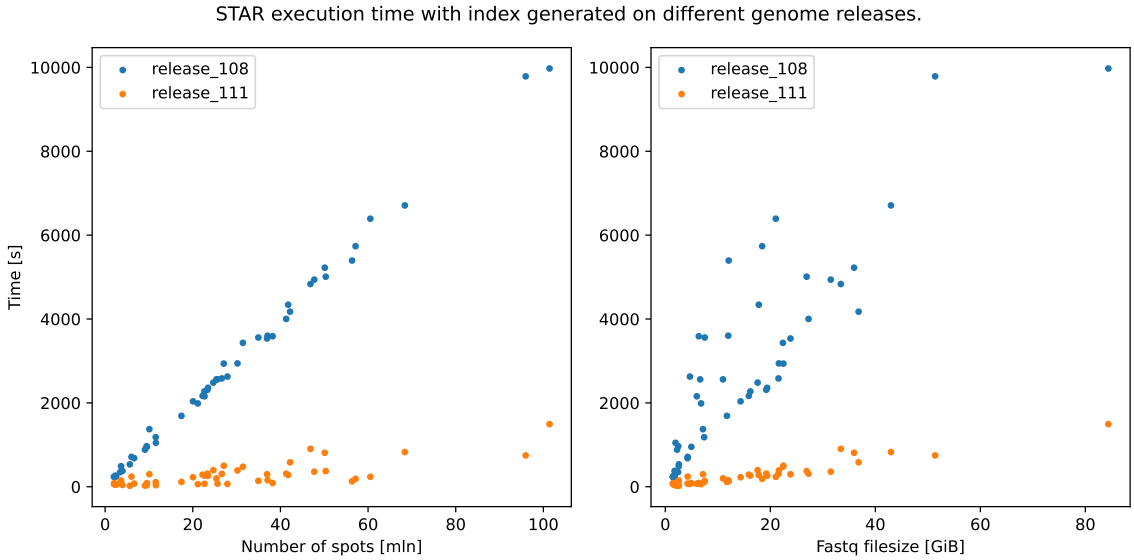



Figure 30: STAR alignment execution time using index generated on Ensembl release 111 and on release 108.

**Spot instances** Using spot instances in AWS can reduce the compute cost by up to 90%, but in our case it is usually about 50-65%. This, however, comes with a downside of possible interruption in contrast to the on-demand model. Each running spot instance can be terminated with a 2-minute notice. Our initial analysis suggests meaningful cost reduction using this execution model. Such analysis takes into account the cost of restarting the worker, starting the computation once again if necessary, and block storage (EBS) costs. This is applicable for both Salmon and STAR pipelines.

With good configuration (e.g. using instance type with low interruption rate) and in an optimal scenario, using spot instances should result in relatively stable computations as shown in Fig. 31.

Here we present results of processing 1000 *SRA* files on r7a.2xlarge instances with the STAR pipeline. During the experiment, only five interruptions occurred and resulted in wasting less than 1% of the total running time among all instances.

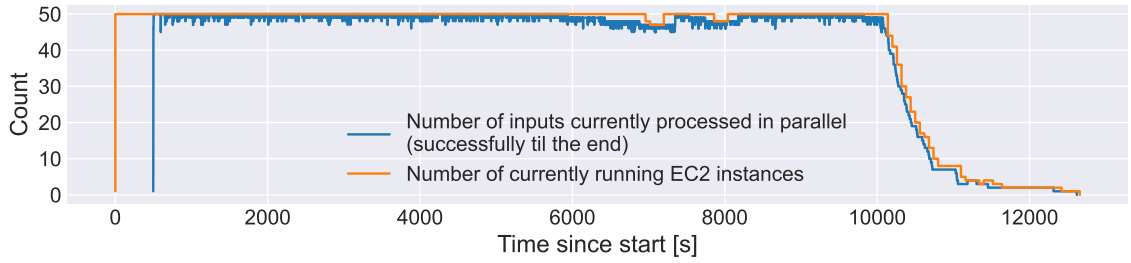



Figure 31: Spot instances usage experiment. Timeline for instances and successfully computed files.

**Cost-efficient instance type for STAR** There are many available instance types on EC2. It is important to choose a type that fulfills all the requirements and is cost-efficient. Loading STAR index into shared memory results in over 29.5GiB of allocated memory. Since STAR is a memory-intensive program, we decide to focus on instances with a higher memory-per-CPU factor (mainly "r" family) with at least 64GiB of RAM. It is possible to use other types with a lower factor and more cores; however, this may require faster block storage and increased under-utilization during other, much less CPU-intensive steps (e.g. *prefetch* step). Selected current generation instance types that fulfill those requirements are compared in Table 20. This table also presents the total cost and time of performing STAR alignment on 50 random *FASTQ* files. We consider only STAR processing time - other pipeline steps make up for 24-31% of total execution time. The results indicate r7a.2xlarge as the fastest and cheapest (on-demand) type. Although, when using spot instances, one should also take into consideration the spot availability of a given type.

Table 20: Cost-efficiency analysis of selected instance types.

| Instance type | vCPU | Cores | RAM [GiB] | On-demand price [h] | Total STAR execution time [h] | Total cost |
|---|---|---|---|---|---|---|
| r6a.2xlarge | 8 | 4 | 64 | $0.4536 | 8.00 | 3.63 $ |
| r6i.2xlarge | 8 | 4 | 64 | $0.5040 | 8.04 | 4.05 $ |
| r7a.2xlarge | 8 | 8 | 64 | $0.6086 | 5.48 | 3.33 $ |
| r7i.2xlarge | 8 | 4 | 64 | $0.5292 | 7.66 | 4.05 $ |

### 5.8.2 KPI-3: Demonstrated resource auto-scaling for batch and stream data processing.

**Scalability** The cloud offers a wide range of services that are easily scalable and fit our use case. Such services include but are not limited to AWS Lambda, ECS running on Fargate, EC2 virtual machines. By using the AutoScalingGroup service on AWS, we can define scaling policies for the processing of the workload. Our solution presents efficient utilization of the resources with good scalability. On HPC, a possible approach for this use case would leverage array-job functionality to run multiple instances of the same job. However, HPC may have limits to the number of available nodes/cores, fair-share policies, queue system, etc. which is not the case in the cloud. By using cloud services and their large compute/storage resources, we can instantly launch multiple instances of workers and therefore, easily parallelize the workload without any queue waiting time (as seen in Fig. 31).

## 5.9 Use-case repositories

The main repositories relevant for this Use Case are listed below:

- Transcriptomics Atlas - `https://github.com/neardata-eu/transcriptomics-atlas-sano`
- Serverless Salmon Pipeline - `https://github.com/neardata-eu/salmonless`
- Federated Learning for Transcriptomics - `https://github.com/SanoScience/neardata-fl-for-transcriptomics`

### 5.10 Discussion and final remarks

The Transcriptomics use case has achieved multiple valuable and measurable optimizations and addressed KPIs described in the project. The total impact of combined optimizations for Transcriptomics Atlas reduced processing cost by two orders of magnitude. Many optimizations are applicable for environments other than cloud (e.g. early stopping on HPC). We successfully integrated and tested Lithops and Dataplug for Salmon pipeline running on AWS Lambda which improved processing time. Also noteworthy addition to Dataplug is the SRA connector. Moreover, we used LithopsHPC to run the pipeline on HPC cluster. Finally, the development related to STAR index access helped us to better understand the memory access pattern of the tool. We have also developed a federated learning framework suitable for transcriptomic and metabolomic experiments.

### 5.11 Future work

Next steps for this use case include expanding the Transcriptomics Atlas dataset with additional tissues. Following the extension, the data will be processed through the established pipeline and the results collected. The finalized dataset will be prepared, analyzed, and then published on e.g. Zenodo.

# 6 Use Case: Pathogen genomics for public health

The UK Health Security Agency (UKHSA) plays a critical role in safeguarding public health across the United Kingdom. As part of its mission, UKHSA routinely sequences a wide range of human pathogens —both viral and bacterial— to generate clinical reports and respond swiftly to outbreaks. The scale of this operation is substantial, with sequencing volumes reaching up to 20,000 pathogens per week, or approximately 1 million per year. This translates to an estimated 0.1–1 petabyte (PB) of raw input data annually, with total data volumes significantly higher when accounting for intermediate and processed outputs. Managing, storing, and analyzing this data efficiently is essential to ensure timely public health interventions and maintain national biosecurity.

Note that while both this use-case and the one in Section 2 nominally target applications of genomics, they are in fact very different — while the former targets Type 2 diabetes, this one focuses on pathogen outbreaks.

## 6.1 Short Description

In this use case, we address two types of efforts:

- **Advances in genomics algorithms for classification, retrieval and comparison of sequences of pathogens**. To support UKHSA's high-throughput pathogen sequencing workflows, research efforts have focused on developing novel genomics algorithms tailored to the agency's infrastructure and operational constraints. Such algorithms aim to optimize the processing of large-scale FASTQ datasets, improve the accuracy and speed of pathogen identification, and enable real-time analytics for outbreak detection. This effort resulted in a novel $k$-mer based method, `KPop` [86]. `KPop` computes complete $k$-mer frequency spectra and embeds them into a low-dimensional space tailored to each dataset, enabling accurate downstream classification, clustering, and nearest-neighbour searches.

- **Technical improvements in the infrastructure needed to perform modern pathogen genomics at scale.** Emerging use cases like real-time pathogen genomics monitoring during outbreaks motivate the exploration of stream processing in genomics [87]. UKHSA is evaluating technologies such as Pravega's byte-oriented streams [88] to ingest FASTQ files and run cutting-edge methods such as `KPop` on them, aiming to enable streaming analytics while maintaining efficient, tiered storage for batch processing. The goal is to allow FASTQ data to be accessed both via Pravega and directly from object storage. In particular, this work evaluates potential advantages of using NEARDATA technologies proposed in D3.2.

## 6.2 Data Spaces

For this use case we focus on the operation at UKHSA of the reference laboratory for a well-known pathogen, *Mycobacterium tuberculosis*, which causes a large burden of infections, many of them presenting several degrees of resistance to treatment, in the UK and worldwide. A clinical pipeline to process tuberculosis (TB) data internally used at UKHSA has been in operation for many years now, resulting in a set of close to 120,000 processed samples. Given that the FASTQ files associated with each sample typically range from 0.5 to 2GB in size, this amounts to a large amount of data, in the region of 100 TB. Due to concerns related to privacy and possible risks that genomic data derived from the host might lead to the sample being identified, this data set is currently being stored locally on UKHSA infrastructure and servers. However, as part of the "cloud-first" approach being currently implemented across the UK government, a move to S3-based storage is being considered for this and other datasets. Hence, it is of paramount importance that a future-proof solution for ingestion from sources in the cloud is considered.

In the meantime, several smaller curated TB datasets from which identifiable information has been eliminated, produced at UKHSA and elsewhere, are currently available in public repositories such as the NCBI Sequence Reads Archive (already described in section 5.2) and can be accessed from AWS S3. These have been used as sources for the experiments presented in this section.

### 6.3 Targeted KPIs

This work is related to the following project KPIs:

- **KPI-2 - Genomics algorithmic improvements**: We validated our novel method, `KPop` [86], on both simulated and real data, including more than 1,000 *Mycobacterium tuberculosis* (TB) genomes and more than 1.28 million SARS-CoV-2 genomes. `KPop` achieves near-perfect classification accuracy and fast query times. By representing genomes as numerical vectors, it bridges genomics and machine learning, offering a foundation for scalable, AI-driven approaches to microbial comparative genomics. In particular, highly parallelizable parts of `KPop` that are an essential, compute-hungry prerequisite to subsequent postprocessing have been reimplemented with NEARDATA technology (see Fig. 32).
- **KPI-1 - Data transfer and storage reduction in FASTQ files**: The integration of the Nexus FASTQGzip streamlet enables significant compression of genomic data during ingestion, achieving up to 3.8× compression ratio while preserving parallel access capabilities. This reduces storage footprint and data transfer costs in batch processing jobs.
- **KPI-5 - Ease of genomics pipeline development**: By abstracting complex infrastructure management and enabling seamless access to genomic data via both stream and object interfaces, FaaStream significantly simplifies pipeline development in NEARDATA. This lowers the barrier for deploying scalable analytics workflows in public health genomics.

### 6.4 Challenges addressed

The UKHSA genomics use case builds upon novel algorithms for sequence analysis and the architectural and methodological innovations introduced in NEARDATA to tackle the unique challenges of processing large-scale datasets relevant to pathogen genomics and human health in a secure, scalable, and efficient manner. The following challenges have been identified and addressed as part of this validation effort:

#### 6.4.1 Quickly process, store, compare and retrieve a large number of pathogen genomic sequences

During everyday operation of clinical services at reference laboratories for specific pathogens (for instance, *Mycobacterium tuberculosis*) a large number of biological samples is received, and such samples subsequently undergo genome sequencing in order to establish whether the sample is known, whether it harbors known markers of resistance to treatment, and whether it is likely to be connected to ongoing or past outbreaks. Each sample generates a relevant amount of raw sequencing data ( 1 GB / sample) that then needs to be processed and analyzed. A large number of samples (in excess of 10,000 / year) is received and processed. In addition to obvious issues related to sheer data size and scalability, one also has to tackle the problem of being able to efficiently store and retrieve the genomic sequence derived from each sample, and to meaningful compare them with one another. In particular, a "relatedness question" is often formulated — is my new sequence related to others already present in the (large) database of known sequences? What are the closest sequences? Are they part of an ongoing outbreak or likely related to the ones which originated another outbreak in the past?

In order to answer such questions, representations of the sequence in terms of specialized data structures do exist. However, such methods are usually cumbersome and/or resource hungry, which makes their implementation cumbersome and their maintenance complex and costly.

#### 6.4.2 Manage redundant and accessible genomic data across stream and object interfaces

The genomics research community is exploring streaming data infrastructures that support both real-time stream processing and long-term object storage [87]. However, most of the legacy genomics workflows still rely purely on file systems or object stores to manage data. Enabling efficient dual-mode access ——where the same genomic data must be accessible both as a stream and as a compressed object— introduces significant complexity. A key challenge is how to reduce enable such

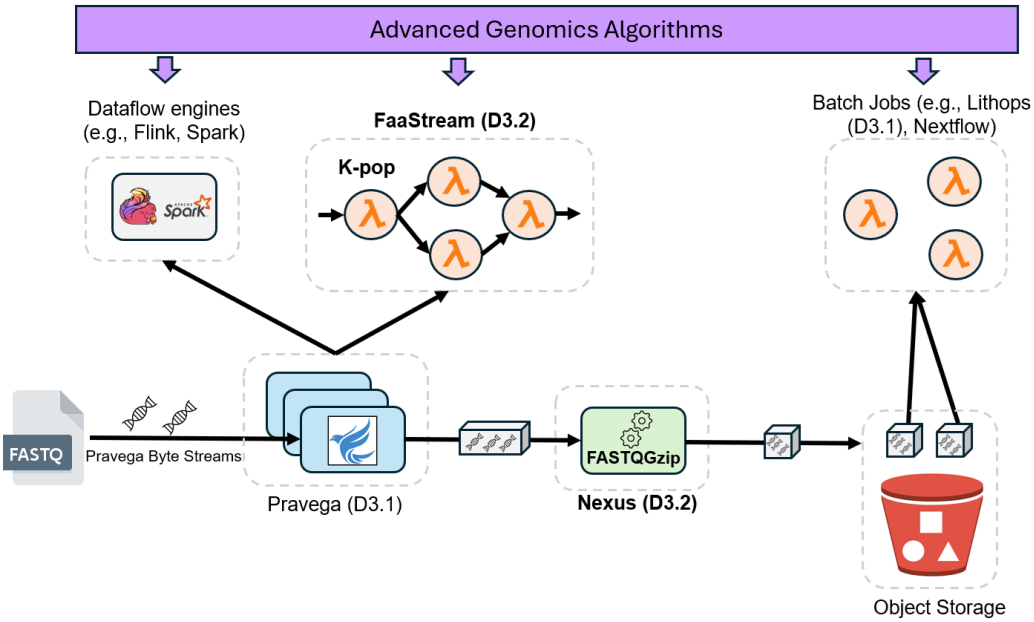



Figure 32: Integration of UKHSA use case components in NEARDATA architecture.

a unified dual access via streaming and object storage APIs. Moreover, there may be multiple data management and data reduction mechanisms that can be explored in the storage tiering process between event streaming systems and object storage. Handling these problems in a unified manner is a challenges we wanted to explore in this work.

### 6.4.3 Reduce complexity in deploying scalable stream analytics for genomics

Executing genomic analytics jobs over —such as `KPop` or other variant prioritization pipelines— typically requires deploying and managing complex data processing engines (*e.g.*, Nextflow, Apache Flink). While these engines offer robust performance, they come with substantial operational overhead, including cluster provisioning, configuration, and maintenance. This complexity becomes a barrier in scenarios where rapid deployment and elasticity are critical, such as in public health genomics. The challenge lies in finding a scalable, low-footprint alternative that can deliver comparable performance without the burden of managing heavyweight infrastructure, especially when workloads are bursty or short-lived.

## 6.5 Integration with NEARDATA architecture components

The UKHSA genomics use case demonstrates a comprehensive integration of NEARDATA architecture components to support scalable, efficient, and flexible processing of large-scale FASTQ datasets. At the core of the data ingestion pipeline is Pravega, which handles byte-oriented FASTQ streams and enables seamless integration with dataflow engines such as Apache Flink and Spark. These engines feed genomic data into Pravega for initial stream-based processing.

However, in addition to traditional cluster-based dataflow engines, the FaaStream framework orchestrates serverless genomic analytics jobs, such as `KPop`, using AWS Lambda functions. FaaStream leverages Pravega's tiered storage and stream-native primitives to dynamically invoke AI-driven genomic workloads, supporting elastic scaling and low operational overhead.

Once ingested, the data is routed through Nexus, a tiered data management mesh that applies streamlet-based transformations. In this use case, the FASTQGzip streamlet compresses subsets of DNA sequences while annotating offsets for parallel access. This enables efficient batch processing without compromising the ability to perform real-time analytics.

Processed data can be consumed by batch job frameworks such as Lithops or Nextflow, which operate on compressed FASTQ chunks, or stored directly in object storage for long-term retention.

This dual-access architecture —stream and object— ensures compatibility with both legacy genomics workflows and modern, cloud-native analytics pipelines. Together, these components form a cohesive and modular architecture that enables UKHSA to manage petabyte-scale genomic data with high performance, flexibility, and resilience, while supporting real-time outbreak monitoring and scalable AI-driven analysis (see section 6.9).

### 6.6 Data connectors developed

The UKHSA genomics use case leverages several data connectors developed within NEARDATA to address the challenges of scalable ingestion, processing, and analytics of large genomic datasets:

- **Nexus FASTQGzip streamlet**: We developed a `ByteStreamlet` that performs GZip data compression on subsets of DNA sequences of an input stream data chunk (*e.g.*, 100K DNA sequences). Each compressed subset of sequences is written as an individual GZip member. Moreover, the streamlet stores the start offset of each GZip member as an object tag. This allows batch jobs to perform parallel access on compressed chunks of tiered stream FASTQ data [89], while regular streaming reads remain unaffected.

- **FaaStream streaming `KPop` job**: This job ingests a parallel stream of FASTQ data from Pravega at the FASTQ record level. It is implemented as a two-stage map-reduce pipeline, where each stage consists of multiple workers (AWS Lambda functions). In the map stage, each worker processes a distinct subset of Pravega stream segments, reading FASTQ records and computing hashes for every subsequence of length $k$ within each genomic sequence. These hashes are then shuffled to the reduce stage, where each reduce worker aggregates the counts for its assigned hashes. The final output is a frequency table of all genomic hashes present in the input stream.

### 6.7 Setup

All the infrastructure-related experiments are conducted in the AWS `us-east-1` region. Unless otherwise specified, AWS Lambda functions are configured with 1,769MB of memory, corresponding to 1 vCPU as per AWS documentation [90]. FaaStream deploys the Pravega cluster and AWS Lambda functions within the same VPC and availability zone to minimize latency and costs. A `m5.4xlarge` EC2 instance, equipped with 16 vCPUs and 64GB of RAM, serves as the client VM for running the FaaStream orchestrator. The Pravega and Nexus clusters are deployed on `i3en.2xlarge` instances, each featuring 8 vCPUs, 64GB of RAM, and 2 dedicated NVMe drives, running Ubuntu 22.04. To simplify provisioning, each instance hosts a Pravega controller, a Pravega segment store, a Bookkeeper instance, and a Zookeeper instance. One NVMe drive is allocated for the Bookkeeper journal and the other for the Bookkeeper ledger and Zookeeper. For long-term storage, Pravega uses an S3 bucket via a VPC S3 endpoint.

Whenever needed, comparisons with HPC implementations were performed on the UKHSA SLURM computer clusters, both the on-premises one (which has about 50 nodes with $\tilde{1}00$ CPU cores and 756 GB of RAM each, and a Lustre filesystem with 3 PB of space) and the contingency cluster implemented on AWS, which has roughly the same basic specifications but can be scaled up elastically to meet demand peaks.

### 6.8 Results and KPIs achieved

Results at a glance are presented in Table 21.

#### 6.8.1 `KPop` workflows to improve pathogen classification, comparison and retrieval

In our paper *`KPop`: accurate and scalable comparative analysis of microbial genomes by sequence embeddings* [86] we introduce `KPop`, a new method for large-scale comparative genomics that combines the precision of full $k$-mer analysis with the scalability of vector-based computation. The motivation behind `KPop` was to overcome the trade-off between resolution and efficiency in existing approaches: while sketch-based tools such as mash are fast, they only sample part of each genome's sequence content, losing information that can be critical when distinguishing closely related strains.

Table 21: Summary of conceptual, architectural and optimization-related KPIs achieved

| Achievement | Benefit |
|---|---|
| Novel algorithms (KPop) | Improved pathogen classification, comparison and retrieval:<br>Almost perfect classification (>98% in all use cases);<br>Retrieval in seconds from a database of >1M sequences;<br>Produces sequence embeddings ready for ML/AI applications |
| Nexus FASTQGzip streamlet | Our Nexus FASTQGzip streamlet offers a 3.8× compression ratio with<br>12.1s parallel processing time, balancing speed and data reduction. Conversely,<br>FASTQ GZip (4.56× compression ratio) cannot parallelize data access<br>and raw FASTQ (≈ 9.4s parallel processing time) provides no data reduction. |
| Serverless stream-based KPop job | FaaStreamis up to 57% faster and up to 65% cheaper than Flink |
| KPop implementation in a Flink-like syntax | Compact code;<br>FaaStream syntax similar to the Flink one. |

Table 22: Summary of KPop validation datasets and classification performance.

| Type | Dataset / Description | Total | Train / Test | Classes | Accuracy |
|---|---|---|---|---|---|
| Simulated | *M. tuberculosis* (synthetic genomes) | 1,000 | 500 / 500 | 10 | **100%** |
| Real data | *M. tuberculosis* complex (SRA) | 1,318 | 668 / 650 | 16 | **98.8%** |
| Simulated | SARS-CoV-2 (synthetic genomes) | 10,000 | 5,000 / 5,000 | 100 | **99.8%** |
| Real data | SARS-CoV-2 (GISAID dataset) | 1.28M | 640K / 640K | 1636 | **98.1%** |

KPop instead computes the complete *k*-mer frequency spectrum for each genome or sequencing dataset and transforms it into a compact, dataset-specific embedding using a dimensionality-reduction procedure. This produces a representation in a lower-dimensional space in which each genome is represented as a numerical vector suitable for clustering, classification, and nearest-neighbor search. The method therefore enables full-resolution, reference-free comparison of microbial genomes at unprecedented scale.

We validated KPop on both simulated and real datasets, including Mycobacterium tuberculosis and SARS-CoV-2 (see Table 22). In simulated bacterial data, it achieved 100% correct classification, compared with 13–33% provided by MinHash-based methods. On real Mycobacterium tuberculosis data, a KPop-based classifier obtained 98.8% accuracy. For viral datasets comprising up to ten thousand genomes, accuracy exceeded 99.8%. Applied to a real-world dataset of more than 1.28 million SARS-CoV-2 genomes, KPop correctly classified 98.1% of sequences at lineage level, while allowing efficient retrieval of nearest neighbors in seconds.

Quite interestingly, the modular nature of KPop allows the same results produced at the beginning of the analysis (which are also the ones we re-implemented with NEARDATA technology) to be used for a variety of purposes. In particular, both classification and nearest-neighbor searches can be performed with slightly different workflows (see Fig. 33). This provides a satisfactory solution to all the problems presented in 6.4.1.

In general, by embedding complete genomic information into vector form, KPop opens the way for AI-driven analyses such as clustering, anomaly detection, and integration with vector databases. We see this as a step toward a new generation of microbial genomics tools that combine accuracy, scalability, and adaptability to modern data volumes.

### 6.8.2 Post-processing FASTQ data from tiered data streams

We compare a simple AWS Lambda job (*i.e.*, DNA sequence count) working on FASTQ files ingested as Pravega streams and processed via our streamlet (see Section 6.6) with two common baselines: FASTQ Gzip and plain FASTQ.

Fig. 34 shows that the compression ratio (4.56×) for the FASTQ Gzip file is the highest, as the

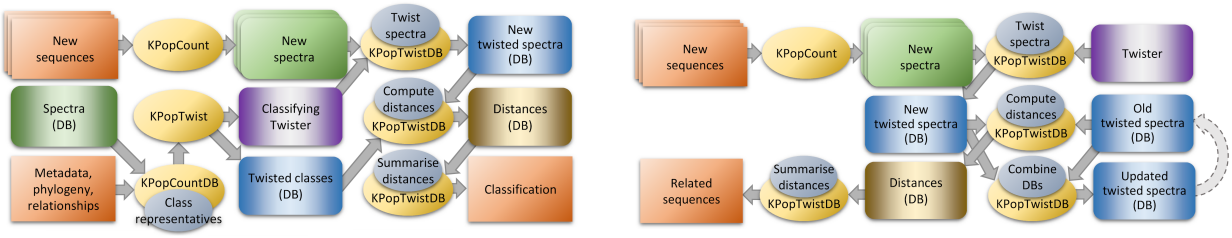



Figure 33: Conceptual designs of a KPop-based classifier and relatedness engine. Comparison shows that most of the computing stages, and all the ones happening upfront, are shared between the workflows (from [86]).

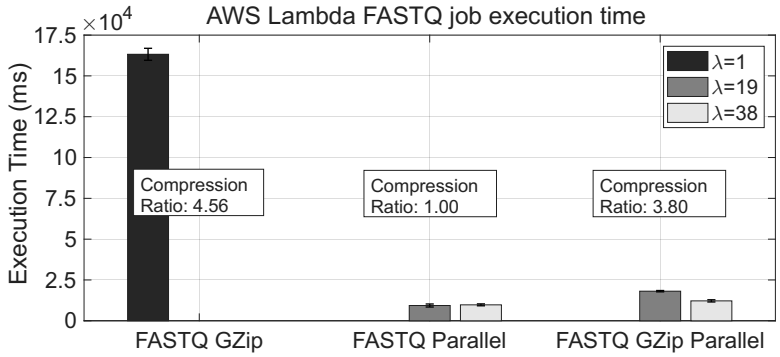



Figure 34: Compression ratio and execution time of AWS Lambda batch jobs on FASTQ data stored via our FASTQ Gzip streamlet versus common baselines.

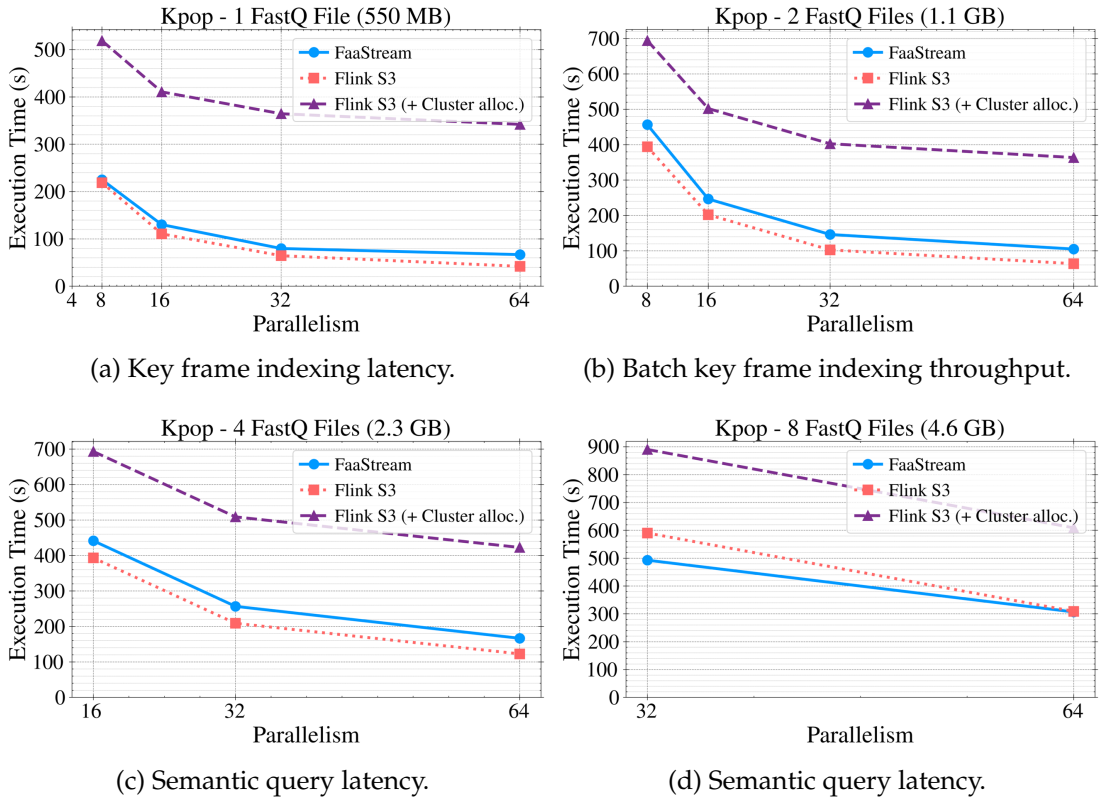
whole file is processed by the GZip compressor. However, compressing a FASTQ file as a single unit prevents multiple lambdas from accessing it in parallel (*i.e.*, $\lambda = 1$). On the other hand, the plain FASTQ file can be processed much faster in parallel ($\approx$ 9.4 seconds), but at the cost of no data compression. Our streamlet provides a compelling trade-off between compression and processing time compared to the tested counterparts (3.8$\times$ compression ratio, 12.1 seconds of processing time for $\lambda = 38$), while being totally transparent to the event streaming system. This allows UKHSA to access FASTQ data either via Pravega or directly processing chunks of tiered stream data via batch frameworks.

### 6.8.3 Serverless stream-based KPop

We compare the performance of our FaaStream KPop job with a baseline Apache Flink implementation that reads FASTQ data from S3 and is deployed via AWS EMR. Fig. 35 shows that FaaStream achieves execution times similar to Flink (on average 23% slower for end-to-end execution), and even outperforms it in certain scenarios (*e.g.*, for the 8 FASTQ files case, FaaStream is up to 16.53% faster). When including cluster allocation and deployment time, FaaStream is substantially faster (on average 57.24%) than Flink, as serverless functions are invoked almost instantly, whereas Flink clusters require several minutes to provision.

In terms of cost, FaaStream is on average 46.58% more expensive than Flink for end-to-end execution, primarily due to the higher per-unit compute cost of AWS Lambda compared to EC2 instances. However, thanks to its elasticity, when accounting for cluster allocation and deployment time, FaaStream becomes on average 65.14% cheaper than Flink (and up to 221.26% in the case of 1 FASTQ file and parallelism 64), as it avoids the cost of idle resources during cluster provisioning.

Listings 1 and 2 show the KPop job implementation for FaaStream and Flink, respectively. As can be seen, FaaStream's programmatic abstractions are similar to those of Flink, facilitating the transition for developers already familiar with Flink's dataflow abstractions. This similarity lowers




(a) Key frame indexing latency.

(b) Batch key frame indexing throughput.

(c) Semantic query latency.

(d) Semantic query latency.

Figure 35: Performance of FaaStream vs Flink executing KPop algorithm on AWS.

Listing 1: KPop job implementation for FaaStream.

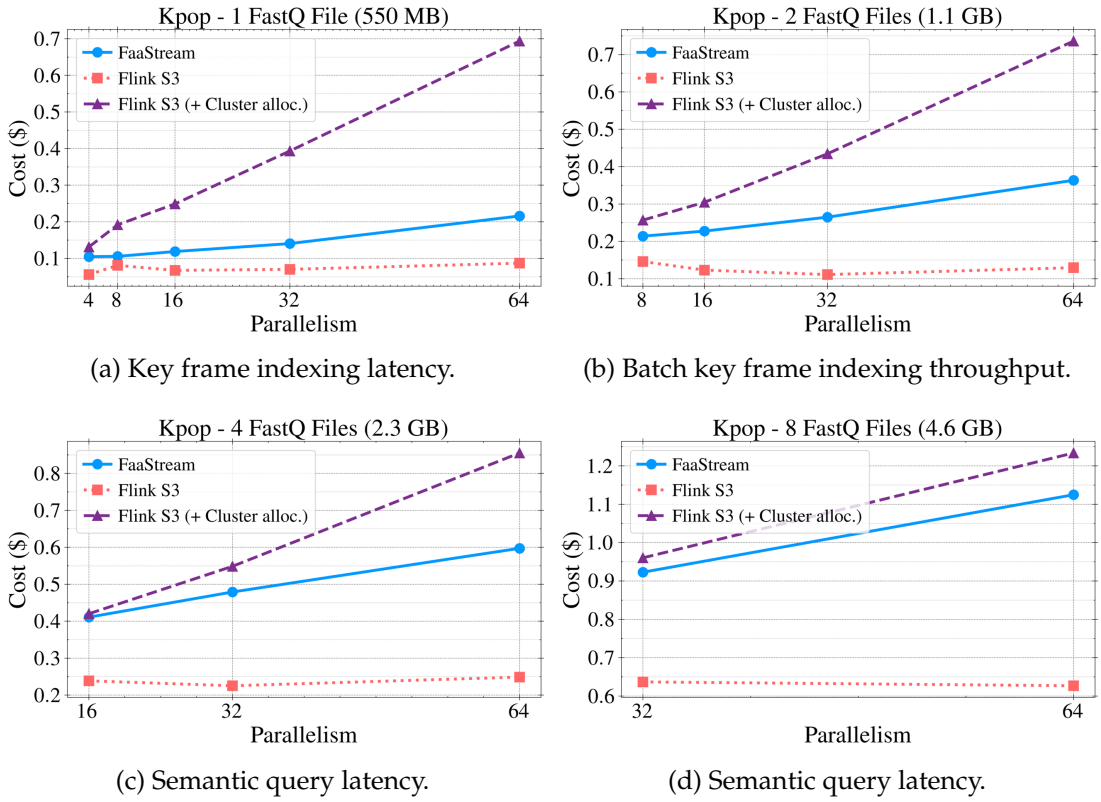
```
Source<FastqMessage> source = Source.fromStream(input);
Pipeline<FastqMessage, HashMap<String, Integer>> pipeline = source
    .flatMap(new KpopMap(k), new KpopHash(), workers)
    .reduce(new KpopReduce(), workers)
    .writeToStream(output);

manager.execute(pipeline, new StaticScalingPolicy());
```

the learning curve, allowing teams to quickly adopt FaaStream for serverless (genomics) analytics without extensive retraining.

### 6.9 How AI is enabled in this use-case

AI capabilities in this use case are enabled through the following distinct architectural contributions that support scalable, efficient, and intelligent processing of genomic data:

- **Providing input to AI-based genomic workflows with KPop-generated embeddings.** Other preexisting popular algorithms for the classification and comparison of microbial genomes (for instance, the class of MinHash-based methods) focus on providing an estimate of distances between different sequences, with the goal of using the generated distance matrix to produce phylogenetic trees with methods such as Neighbor Joining. While this is sufficient for some applications, it is not helpful with more advanced applications based on deep-learning (or machine-learning) which usually need data points as vectors in some abstract space ("embeddings") as a starting point. Given this, KPop brings a distinctive novelty to the field by being able to generate embeddings as a key point of its design, which represents a significant strength.




(a) Key frame indexing latency.

(b) Batch key frame indexing throughput.

(c) Semantic query latency.

(d) Semantic query latency.

Figure 36: Cost of FaaStream vs Flink executing the KPop algorithm on AWS.

In addition, KPop can do that in a way which is optimized for each specific dataset, which significantly improves the quality of the generated embeddings at all scales, both for close and distant sequences (see Fig. 37). Also note that strictly speaking, the method used in KPop can be applied to general text, not just DNA or proteins, and the operations of encoding text and $k$-mer extraction can be performed in different ways. These parts of the method have been completely re-implemented and generalized to support advanced features in October 2025.

- **Enabling seamless AI-driven genomic workflows across stream and object interfaces.** AI enablement in this use case is driven by the integration of Nexus streamlets into a dual-access data architecture that supports both streaming and object-based interfaces. This design allows diverse AI and ML engines —including serverless functions, batch pipelines, and inference frameworks— to operate seamlessly over genomic datasets without requiring format conversion or data duplication. The streamlets play a critical role in managing and transforming

Listing 2: KPop job implementation for Flink.

```
DataStream<FastqMessage> source = env.createInput(
    new FastqInputFormat(new Path(input)));
DataStream<HashCount> pipeline = source
        .flatMap(new HashFunction(k))
        .keyBy(HashCount::getHash)
        .sum("count");

pipeline.sinkTo(sink);
env.execute("KPOP␣Flink␣Job");
```

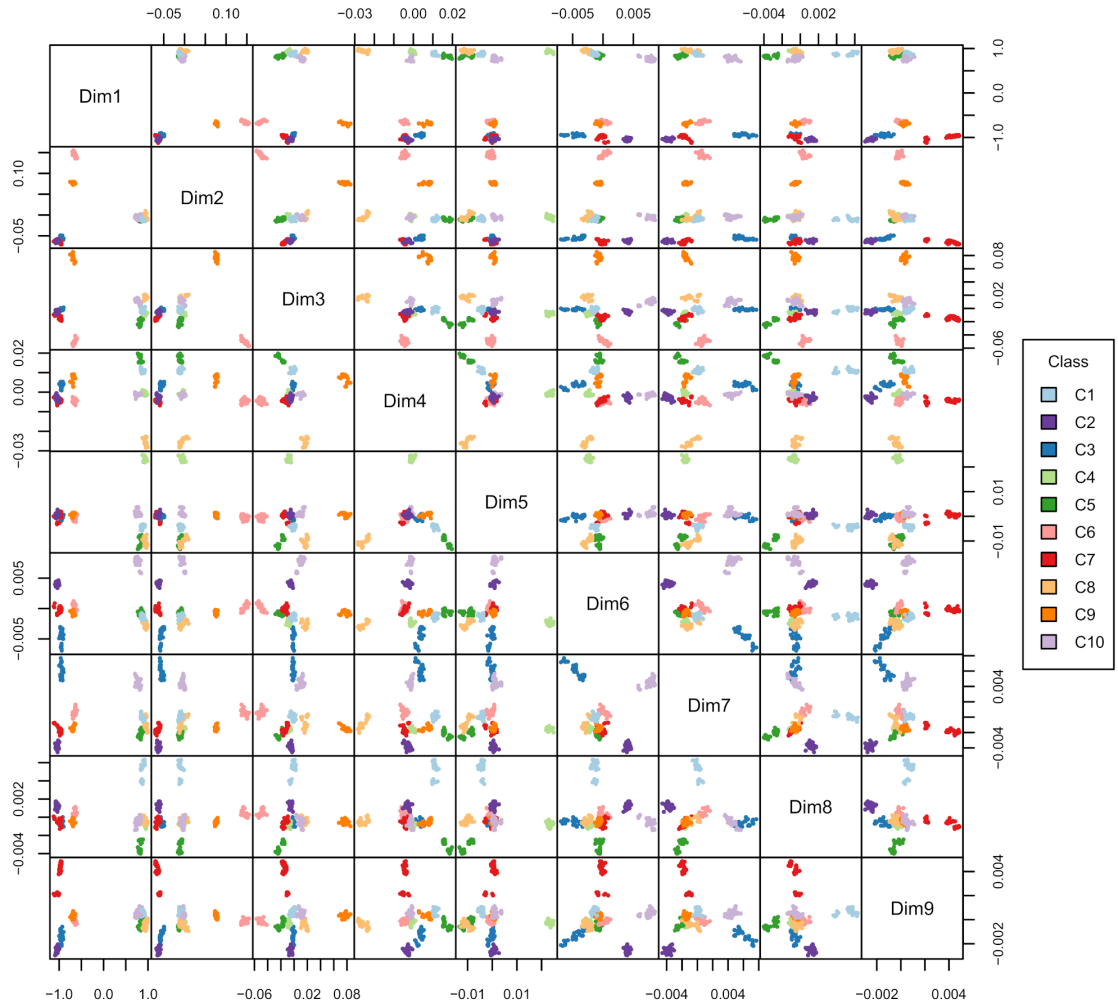



Figure 37: Visualization of multi-dimensional embeddings generated by KPop in a test example (from [91]).

FASTQ data during ingestion, applying windowed Gzip compression to reduce data volume while preserving parallel access capabilities. This enables efficient execution of AI-driven analytics across compressed genomic data, supporting scalable model training and inference. By combining data reduction with flexible access paths, the architecture ensures that AI workloads can be deployed dynamically and efficiently across heterogeneous infrastructures.

- **Serverless AI execution for genomic pipelines via FaaStream.** To support scalable execution of genomic analytics jobs like KPop, we leverage FaaStream —a serverless framework built on tiered data streams. FaaStream enables the orchestration of AI workloads using cloud functions, eliminating the need for complex cluster deployments. In this use case, AI is enabled through the dynamic invocation of functions that process genomic data in parallel, using stream-native primitives such as shuffling and stateful coordination. This architecture allows for rapid deployment and elastic scaling of AI pipelines, making it suitable for bursty or short-lived genomic tasks.

### 6.10 Use-case repositories

The UKHSA genomics use case builds upon the KPop method and algorithms as well as core components of the NEARDATA architecture, including Nexus and FaaStream. The source code and doc-

umentation for these components are publicly available to support reproducibility and community adoption. Specifically:

- The KPop suite of applications are available at [91]. A set of user-friendly workflows based on Nextflow can be found at [92].
- The Nexus streamlet framework for tiered data management and the FaaStream serverless orchestration engine can be accessed at [93] and [94], respectively.

### 6.11 Discussion and final remarks

The UKHSA genomics use case demonstrates the applicability of NEARDATA's architectural innovations to real-world, high-throughput public health genomics. By integrating cutting edge new methods, stream-native data connectors and serverless analytics frameworks, UKHSA is able to address key challenges in managing large amounts of pathogen samples relevant to public health. Sequencing data derived from them can be efficiently and scalably processed, retrieved and compared; this goes all the way from the original FASTQ files to the high-level representation of genomic sequences produced by our novel method `KPop`. These results pave the way for a robust, scalable system where biological samples can be automatically turned into an AI-friendly format ready for further downstream processing.

The Nexus FASTQGzip streamlet offers a compelling balance between compression and parallel access, enabling efficient ingestion and analytics without compromising performance. Meanwhile, the FaaStream framework allows for elastic, serverless execution of genomics pipelines such as `KPop`, reducing operational overhead and enabling rapid deployment in outbreak scenarios. On top of that, stream-based workflows allow for real-time, per-sample tracking of the current state of the computation; this makes it possible to deploy dashboards to which selective access can be provided, which is paramount in an inherently federated system such as UKHSA where many entities —hospitals, pathogen reference labs, epidemiologists and decision makers— work together to improve human health.

These results validate the feasibility of stream-based genomics infrastructures for national-scale pathogen surveillance and open the door to further innovations in real-time analytics, AI-driven variant prioritization, and secure data sharing across health institutions.

### 6.12 Future work

Building on the validation efforts carried out in the UKHSA genomics use case, future work will focus on expanding the integration of NEARDATA technologies into operational public health infrastructures. One key direction is the deployment of real-time outbreak monitoring systems that leverage stream-based ingestion and analytics to enable faster sample processing and response. This may involve extending the use of Pravega and Nexus to support continuous sequencing pipelines and dynamic analytics workflows.

Another important area of development is the enhancement of the streamlet functionality to support additional genomic formats such as CRAM and BAM, which are widely used in clinical genomics. This will improve compatibility with existing tools and broaden the applicability of the NEARDATA architecture.

Finally, improvements in metadata tagging and indexing for sequencing data will be explored to support advanced search, retrieval, and auditability across tiered storage systems. This nicely supplements the native capabilities offered by stream-based processing —namely the provision of real-time updates about sample processing and selective access control— to different actors of a complex, federated public health system such as UKHSA, in which many actors with different roles interact and require access to sensitive information. Together, these technical enhancements will contribute to a more robust, scalable, and intelligent genomics infrastructure for national biosecurity and public health.

# 7 Use Case: Metabolomics

## 7.1 Short Description

The starting point for this use case is the METASPACE platform, which provides an open ecosystem for processing and sharing metabolomics datasets. METASPACE offers end-to-end functionality, from raw imzML/ibd files to annotated molecular features and visualization-ready outputs. However, when deploying this workflow at scale within NEARDATA, we encountered three main challenges that required methodological and architectural innovations.

First, the challenge of data fragmentation. Public metabolomics data are distributed across multiple repositories, each with its own access mechanisms, metadata standards, and usage constraints. To address this, we developed a federated Data Space that unifies four key repositories: Metaspace, AWS Open Registry, MetaboLights, and Metabolomics Workbench. This integration is enabled through dedicated connectors, with DataCockpit supporting AWS and Metaspace, DATOMA integrating MetaboLights and Metabolomics Workbench, and PyRun leveraging DataCockpit to execute pipelines directly on this federated Data Space. Together, these components mitigate fragmentation and ensure datasets are delivered in a standardized, partitioned form, ready for large-scale analysis.

Second, the challenge of scalability. The metabolomics pipeline mixes stages with heterogeneous compute and I/O profiles, making fixed clusters either underutilized or overloaded. We addressed this by adopting a fully serverless execution model, allowing concurrency to scale elastically per stage. We conducted two experiments to stress different pipeline dimensions: (i) fixing the dataset size (1.8 GB) while increasing the reference database from 500 to 12 000 formulas, and (ii) fixing the database at 12 000 formulas while scaling dataset size (1.8 $\rightarrow$ 7.0 GB). We further optimized ingestion by migrating from the sequential `load_ds` to the distributed `load_ds_parallel` and tuning partition size with DataPlug + DataCockpit. These optimizations significantly improved performance while maintaining reproducibility.

Finally, the challenge of confidential computing. Metabolomics data often involve sensitive biomedical information, requiring protection not only at rest and in transit but also in use. We therefore explored the application of confidential computing to the metabolomics pipeline, leveraging Trusted Execution Environments (TEEs) to execute unmodified code securely, thereby extending the Data Space and pipeline execution model into a privacy-preserving domain.

## 7.2 Platforms Description

The development of a federated Data Space for metabolomics is enabled by three key technologies within the NEARDATA ecosystem: DataCockpit, DATOMA, and PyRun. Each plays a complementary role, addressing data access, application availability, and execution scalability.

- **DataCockpit** is a lightweight orchestration and browsing tool that connects directly to object stores such as AWS Open Registry and Metaspace. Built on top of DataPlug, it enables on-the-fly dataset partitioning and prepares data for large-scale parallel execution. By combining dataset discovery with automatic chunking, DataCockpit transforms raw, heterogeneous repositories into standardized, benchmarkable inputs for serverless pipelines. Implemented as a reusable Jupyter notebook widget, it allows researchers to interactively explore datasets, configure partitioning, and orchestrate execution from within their analysis environment. This capability is particularly valuable in the metabolomics use case, where large imzML/ibd files can be partitioned into optimal chunks (e.g., 50 MB) and executed efficiently across thousands of cloud functions.

- **DATOMA** is conceived as an app store for metabolomics that integrates MetaboLights and Metabolomics Workbench repositories. It provides researchers with direct access to curated datasets while offering pre-configured applications that can be executed without deep technical expertise. Within the Data Space, DATOMA plays the role of lowering the entry barrier for non-expert users, linking high-quality, FAIR-aligned datasets with reusable workflows. By

abstracting technical details of data access and format heterogeneity, DATOMA fosters reproducibility and community adoption of metabolomics applications.

- **PyRun** functions as a marketplace of pipelines across multiple scientific domains, with metabolomics as a key application area. It provides a managed execution environment that integrates seamlessly with DataCockpit, enabling reproducible and scalable workflows over the federated Data Space. PyRun automates provisioning, scaling, and monitoring on cloud infrastructures, reducing operational complexity for both expert and non-expert users. By positioning pipelines as modular, shareable assets, PyRun broadens the reach of NEARDATA and facilitates cross-domain adoption of serverless analytics.

Together, these three components form the operational backbone of the metabolomics use case: DataCockpit standardizes and partitions data, DATOMA bridges repositories and applications, and PyRun orchestrates scalable pipeline execution.

### 7.3 Data Spaces

The metabolomics Data Space we developed brings together the most relevant open repositories, each with its own protocols, APIs, and access constraints:

Four representative datasets are considered for the experiments:

- **Metaspace:** specialized in mass spectrometry imaging (MSI).
- **AWS Open Registry:** cloud-based open datasets hosted on Amazon Web Services, supporting large-scale distributed analysis.
- **MetaboLights:** curated FAIR datasets maintained by EMBL-EBI, closely linked to scientific publications.
- **Metabolomics Workbench:** an international reference repository for standardized metabolomics studies, especially in biomedical research.

This heterogeneous landscape is made usable through our integration layer, which transforms it into a federated and standardized Data Space. DataCockpit currently supports AWS Open Registry and Metaspace, automatically partitioning datasets via DataPlug for parallel execution. DATOMA extends the ecosystem by integrating Metaspace and MetaboLights into an app store for metabolomics, providing FAIR datasets alongside ready-to-use applications. Finally, PyRun acts as a pipeline marketplace across multiple domains, including metabolomics, and leverages DataCockpit to execute workflows reproducibly over the federated Data Space.

Through this infrastructure, we accessed and processed representative MSI datasets for our scalability experiments — Brain02 Bregma +1.42 (50 MB), CT26 xenograft (1.8 GB), Mouse brain test434x902 (3.9 GB), and X089-Mousebrain (7.0 GB) — as well as centroid libraries of 500 and 12 000 formulas. The federated design ensured that these heterogeneous datasets could be delivered in a partitioned and ready-to-process form, enabling optimal performance in serverless pipelines.

### 7.4 Targeted KPIs

This work is related to the following project KPIs:

- **KPI-3 - Resource auto-scaling (data-driven orchestration)**: The pipeline runs fully serverless on Lithops, so concurrency adapts to stage demands. Compute-intensive, database-dependent steps (e.g., centroid matching/processing) fan out to hundreds of workers, while lighter or sequential steps run with few or single workers, producing clear "concurrency waves" across the workflow. As input size or database complexity grows, the system scales the number of invocations but maintains stable per-Lambda CPU/memory profiles, indicating elastic right-sizing rather than fixed over-provisioning.

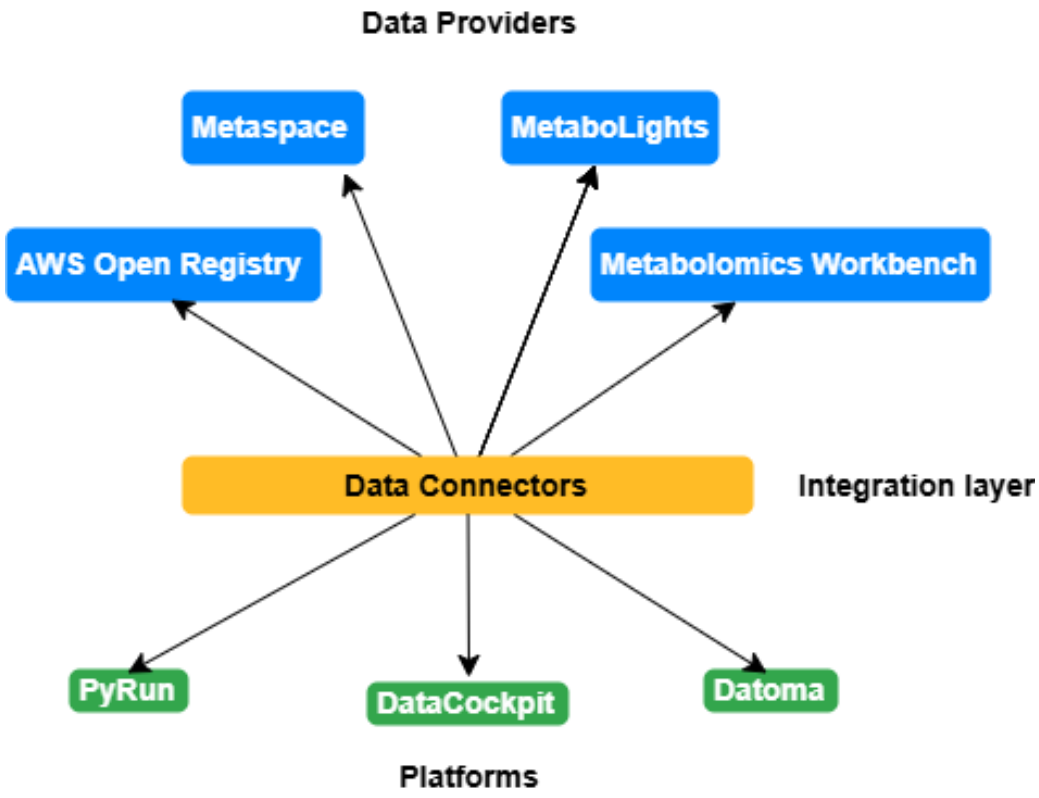



Figure 38: Federated metabolomics Data Space integrating providers, connectors, and platforms (DataCockpit, DATOMA, PyRun).

- **KPI-1 - ETL performance: throughput/transfer reduction**: We target ETL efficiency by replacing the original `load_ds` (a single-Lambda, sequential loader) with the distributed `load_ds_parallel`. The former concentrated parsing, sorting and partitioning in one function, which limited peak throughput and, in practice, prevented execution on larger datasets due to memory and wall-time constraints. The parallel loader applies a distributed sort pattern. We then tuned `upload_ partitions` using Dataplug + DataCockpit, sweeping chunk sizes and selecting 50 MB as a robust operating point. Compared to a 130 MB baseline, 50 MB reduced ingestion time.
- **KPI-5 - Simplicity and productivity**: Running on PyRun (managed serverless Python) and Lithops (FaaS orchestration) removed setup and provisioning overheads, enabling rapid, iterative experimentation (e.g., chunk-size sweeps, parameter ablations) with built-in telemetry for CPU, memory, disk, network and timelines. The FaaS model further improves developer productivity by abstracting infrastructure management (autoscaling, failure handling, packaging), so effort focuses on stage logic and measurement rather than cluster configuration.
- **KPI-4 - High levels of data security and confidential computing validated using TEEs**: The pipeline execution in TEEs of ported applications into confidential computing and other protection mechanisms.

### 7.5 Challenges addressed

This use case exposed several methodological and engineering challenges stemming from the heterogeneity of the metabolomics pipeline, the characteristics of imaging MS data (imzML/ibd), and the choice of a fully serverless execution model. Below we summarize the main challenges and how we addressed them.

*Pipeline heterogeneity and fit-for-purpose parallelism*: The pipeline mixes stages with very different compute and I/O profiles. Database-dependent steps (e.g., centroid matching and centroid-segment processing) are CPU and memory-intensive and benefit from massive fan-out, whereas dataset-dependent steps (e.g., ingestion, per-segment merges) are I/O-bound and sensitive to object-storage behavior. Besides, there are some sequential steps (e.g., bounds estimation, database preparation) which limit overall speedups (Amdahl's law). Because fixed-size clusters tend to be either underutilized during light phases or insufficient during bursts, we used a serverless, stage-oriented design where concurrency is granted on demand and remains low otherwise, avoiding idle capacity and simplifying cost/performance trade-offs.

*Data ingestion bottleneck and scaling limits*: The original `load_ds` ingested an entire imzML/ibd pair inside a single function, which directly limited the dataset size to a certain point. We replaced our executions with `load_ds_ parallel`, a distributed ingest that estimates m/z bounds, performs a map over partitions with windowed reads from the ibd file, and reduces into per-segment outputs, permitting the user to execute the serverless version of this pipeline with larger dataset sizes.

*Resource allocation under FaaS constraints*: Since the number of chunks used for dataset ingestion is configurable by the user, this becomes a tuning problem: small chunks reduce per-task memory and speed up seeks but increase overhead (more invocations, more small writes); large chunks amortize overhead but inflate serialization time and memory. The optimal point depends on dataset size and format, storage throughput and runtime packaging. We made allocation evidence-driven using the DataCockpit tool implemented on Pyrun, which uses Dataplug for dataset partitioning and permits the user to benchmark a given function for finding the optimal chunk size. We extracted the logic of the `upload_partitions` stage into a benchmarkable function, swept chunk sizes and selected a setting that minimized elapsed time. Empirically, a 50 MB chunk size provided the best time-to-ingest across our datasets when compared with a 130 MB baseline, and the same optimum held when validating the stage in isolation.

*Data Space heterogeneity and provider constraints*: In parallel, we faced challenges from the diversity of data providers (Metaspace, AWS Open Registry, MetaboLights, and Metabolomics Workbench). Each exposes different protocols and APIs, and several impose restrictions such as bandwidth limits or selective access policies. We addressed this by developing dedicated data connectors and integrating them into DataCockpit and DATOMA, enabling standardized and partitioned access to heterogeneous datasets. While this mitigates fragmentation, full interoperability and guaranteed openness remain open challenges.

*Pipeline execution ported to confidential computing*: Keeping the secrecy of data throughout the computing continuum is essential in cases where leaking PII (Personally Identifiable Information is any information related to an identifiable natural person) can expose a person's private information or even endanger them. To counter this issue, Metaspace has been ported to SCONE and executed in TEE (Trusted Execution Environment is piece of hardware which allows secure processing) and supported by other mechanisms of confidential computing.

### 7.6 Data connectors developed

To enable this federated access, we developed a set of data connectors that link our components with the repositories in the Data Space:

- AWS Open Registry → connector available in DataCockpit.
- Metaspace → connector available in both DataCockpit and DATOMA.
- MetaboLights → connector integrated into DATOMA.
- Metabolomics Workbench → connector integrated into DATOMA.

These connectors eliminate the need for users to work directly with heterogeneous APIs, offering a unified access point to data. Combined with DataPlug, they transform diverse datasets into partitioned, analysis-ready blocks suitable for parallel and reproducible processing.

Table 23: Integration matrix between Data Providers and Platforms in the metabolomics Data Space.

| Data Provider | DataCockpit | DATOMA | PyRun |
|---|---|---|---|
| Metaspace | X | X | via DataCockpit |
| AWS Open Registry | X | | via DataCockpit |
| MetaboLights | | X | |
| Metabolomics Workbench | | X | |

### 7.7 How AI is enabled in the use-case

In the spatial metabolomics use case, artificial intelligence is the core enabler of metabolite identification within the METASPACE platform. Building on the work described in the previous deliverable, NEARDATA D5.1, the rule-based expert system originally deployed in METASPACE has been replaced in production by a machine-learning (ML) pipeline that learns from features computed directly from imaging mass spectrometry data and associated metadata. As reported in D5.1, this shift from rules to learning consistently improved scientific performance across 720 test datasets: the average PR-AUC rose from 0.25 to 0.45, the MAP increased from 0.3 to 0.5, and the number of metabolites identified at 10% FDR grew by 10–100 per dataset. These results demonstrate that AI not only reproduces expert judgments but systematically recovers additional biochemically plausible signals—the so-called "dark matter" of spatial metabolomics.

The AI pipeline is engineered for extreme-data operation by running near the data in a serverless architecture orchestrated by Lithops, as detailed in D5.1. Feature extraction, model inference, and FDR estimation are expressed as fine-grained, data-driven tasks that auto-scale across heterogeneous inputs. D5.1 documented that end-to-end runtimes for the ML-based FDR stage remained in the 1.7–4.58 s range with only minor dependence on dataset size, validating scalability under increasing loads. Although the ML workflow is about 1.5–2× slower than the legacy rule-based path due to the larger set of molecules it confidently annotates, the execution cost remains essentially unchanged (101–110% of the baseline), thanks to the elasticity and parallelization afforded by serverless execution.

Besides, the output of the metabolomics pipeline—partitioned and processed ion images derived from datasets in **METASPACE**—can be directly reused as input for AI models addressing the off-sample recognition problem. In particular, convolutional neural networks such as ResNet50, as developed in *OffsampleAI* [95], can classify ion images into *on-sample* and *off-sample* categories.

By exposing ion images in a standardized and ready-to-use format through the federated Data Space, our pipeline eliminates the need for manual preprocessing and enables AI models to operate at scale. The combination of **DataCockpit + DataPlug** ensures that large MSI datasets (imzML/ibd) are efficiently partitioned, while **PyRun** orchestrates the execution of these AI-enabled workflows across cloud infrastructures. This integration demonstrates how data preparation and large-scale processing feed directly into machine and deep learning methods, strengthening the interpretability and automation of metabolomics analyses.

### 7.8 Integration with NEARDATA architecture components

The metabolomics Data Space and its supporting tools have been designed to align seamlessly with the NEARDATA architecture, ensuring that data access, preparation, and scalable execution are integrated into the broader ecosystem. Their contributions map onto the architecture's layers as follows:

- Data Plane:
  - Data Connectors and Partitioning: The connectors we developed for Metaspace, AWS Open Registry, MetaboLights, and Metabolomics Workbench extend NEARDATA's Connector Engine capabilities. Through DataPlug, large and heterogeneous metabolomics

datasets are automatically partitioned and converted into analysis-ready chunks, enabling efficient downstream processing.

  - DataCockpit serves as an interactive interface within the Connector Engine, allowing users to browse and select datasets across repositories, trigger partitioning via DataPlug, and orchestrate parallel ingestion. This directly supports the NEARDATA goal of making heterogeneous data sources accessible through a unified entry point.
  - Metaspace, already present in NEARDATA as a Data Catalog, is further leveraged in our integration as a metadata-driven entry point for spatial metabolomics datasets, connecting discovery with automated ingestion and scaling.

- Analytics Layer:
  - Lithops acts as the serverless execution engine that consumes partitioned metabolomics data at scale. When datasets from the federated Data Space are prepared via DataPlug and DataCockpit, Lithops orchestrates massively parallel function execution close to storage, ensuring elastic and efficient processing.
  - The scalability experiments we carried out (varying dataset sizes and database formulas) directly benefited from this integration, as partitioned data streams generated through NEARDATA's Data Plane fed seamlessly into Lithops-based analytics pipelines.

- Control Plane:
  - PyRun provides orchestration across cloud infrastructures, integrating with Lithops, Ray, and Dask to deploy metabolomics pipelines. Within NEARDATA, PyRun contributes to Confidential and Federated Orchestration, automating provisioning, scaling, and cleanup, while exposing a unified web IDE for workflow execution.
  - DATOMA, although primarily conceived as an app store for metabolomics, complements the orchestration perspective by enabling discoverability and deployment of FAIR-compliant metabolomics applications. Together with PyRun, it closes the loop between data access, pipeline execution, and application delivery.

By embedding the metabolomics Data Space within NEARDATA's architectural framework, we demonstrate how federated data access, automatic partitioning, and scalable execution can be achieved in a domain-specific context. This integration not only validates NEARDATA's abstractions in the metabolomics use case but also highlights the generalizability of the architecture to other scientific domains.

### 7.9 Results and KPIs achieved

#### 7.9.1 Pipeline Execution with Different Database Sizes

To isolate the impact of database-dependent computations, we fixed the input dataset to the 1.8 GB *CT26_xenograft* sample and executed the full pipeline twice, varying only the size of the metabolite reference. We used the distributed dataset loader `load_ds_parallel` with 15 initial chunks and 8 m/z segments (parameters chosen to balance parallelism and per-task overhead). One run used a minimal database of 500 adduct formulas and the other a **larger database of 12000 formulas**. By holding all other factors constant, we can directly compare how these database-dependent stages dynamically scale with database size.

When comparing the 500-formula and 12000-formula reference runs, the database-dependent stages exhibit the most dramatic scaling behavior. In **Fig. 39**, `calculate_peaks_chunk`, `get_segm_stats`, `save_png_chunk` and `process_centr_segment` each spin up only a few dozen Lambdas on the small database but surge to hundreds of concurrent invocations with the larger one (peaking at aprox. 70, 270, 270 and 30 active calls, respectively). As shown in **Fig. 40a**, this increase corresponds to a matching rise in total function calls, while non-database-dependent stages remain at constant invocation counts.

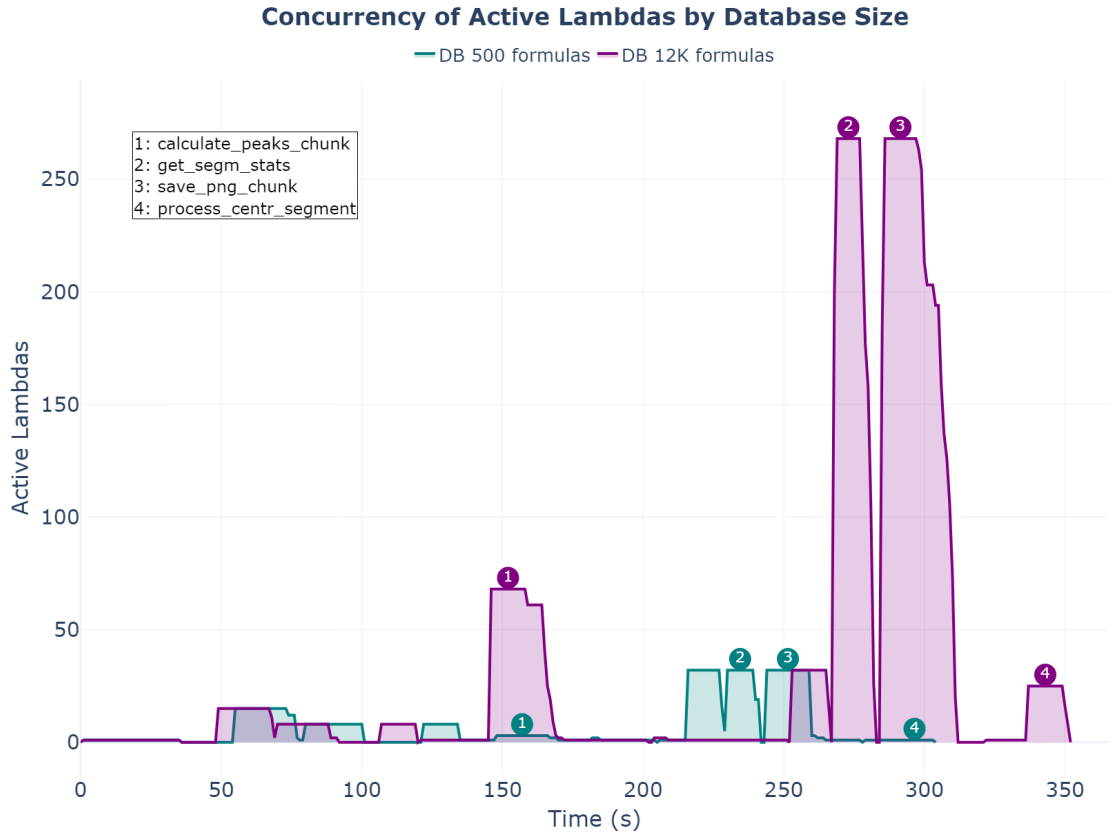



Figure 39: Concurrency of active AWS Lambda functions over time for the two database sizes. The plot highlights the `calculate_peaks_chunk`, `get_segm_stats`, `save_png_chunk` and `process_centr_segment` functions to illustrate how their level of parallelism increases with database size.

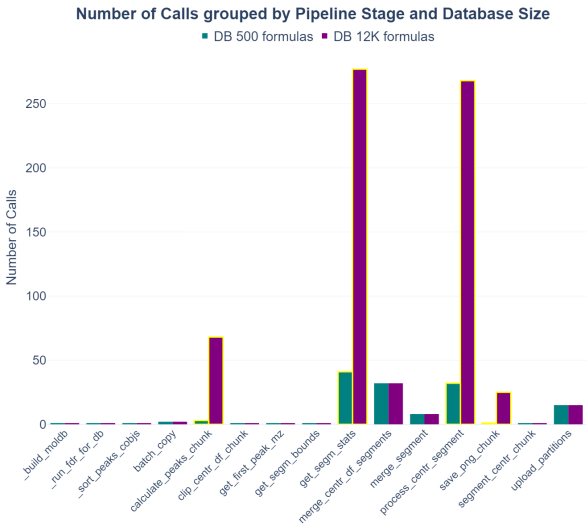
Table 24: Setup of the two 1.8 GB pipeline runs with different reference database sizes

| Dataset | Size | Reference DB | # Chunks | # Segments |
|---|---|---|---|---|
| CT26_xenograft | 1.8 GB | 500 formulas | 15 | 8 |
| CT26_xenograft | 1.8 GB | 12 000 formulas | 15 | 8 |

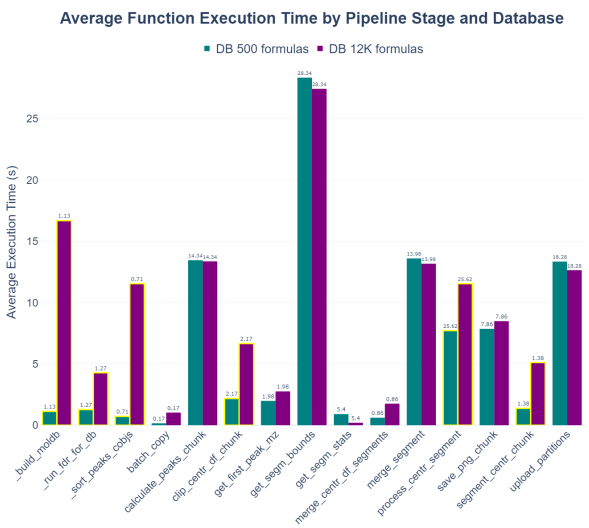
**Figure 40b** offers an insightful illustration of the serverless architecture's contribution: due to the elasticity of serverless components, resource-demanding stages (such as `process_centr_segment` or `get_segm_stats`) maintain very similar execution times across different database sizes. This is because cloud functions enable the immediate scaling of resources, meeting stage requirements with very low latency. On the other hand, as expected, purely sequential steps such as `_build_moldb` suffer the greatest slowdown under the larger databases.

For **Figure 41**, we filtered the data to include only the database-dependent stages—`calculate_peaks_chunk`, `get_segm_stats`, `process_centr_segment`, and `save_png_chunk`. This focus allows us to illustrate how these stages maintain effective CPU and memory provisioning as the reference database grows in size and complexity, reflecting stable utilization profiles under increased load.

Distributing the same total work across many more Lambdas when using the larger database yields similar per-function CPU load (**Figure 41a**): We see the average utilization drop only slightly from 10.3% to 8.1%, and the peak from 35.1% to 31.8%. This is because each invocation spends proportionally more time waiting on I/O or synchronization.

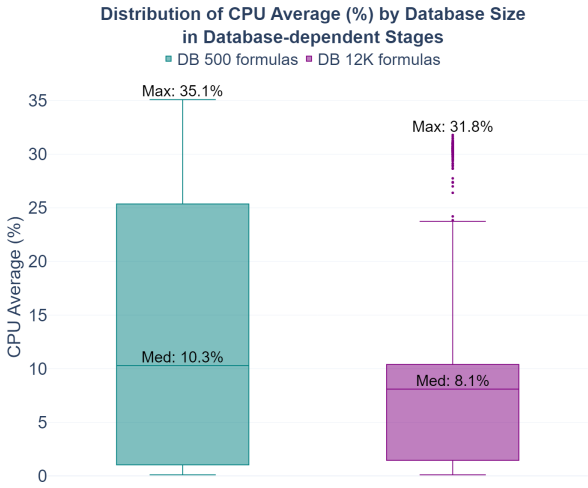



(a) Total number of function calls per stage for each database size.

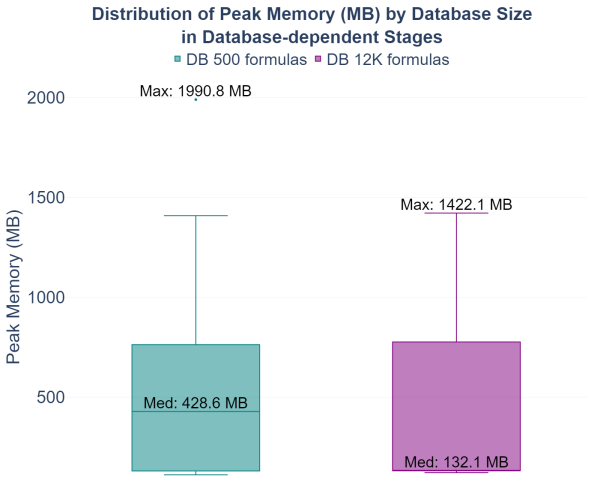



(b) Average execution time per pipeline stage across database sizes.

Figure 40: Comparison of function call numbers and stage latencies for two increasing database sizes.




(a) Distribution of average CPU utilization (%) per database size in selected stages.




(b) Distribution of peak memory usage (MB) per database size in selected stages.

Figure 41: Comparison of resource usage distributions across different database sizes in database-dependent stages: `calculate_peaks_chunk`, `get_segm_stats`, `process_centr_segment` and `save_png_chunk`: (a) average CPU utilization, (b) peak memory usage.

The memory footprint per Lambda remains essentially unchanged (**Figure 41b**). These observations consistently highlight the substantial scalability inherent in serverless architectures.

Together, the comparison demonstrates that our serverless `lithops` pipeline deployment effectively leverages broad parallelism to contain latency growth and balance resource demands as database complexity increases.

### 7.9.2 Pipeline Execution with Different Dataset Sizes

We ran three large-scale experiments (all using the database of 12000 formulas used before), as shown in table 25. We enabled the `load_ds_parallel` loader in each run, tuning the number of initial chunks (`ds_n_chunks`) and the number of m/z segments (`ds_n_segms`).

Table 25: Summary of Executions

| Dataset | Size | Description | # Chunks | # Segments |
|---|---|---|---|---|
| CT26_xenograft | 1.8 GB | Xenograft tumor section | 15 | 8 |
| Mouse brain test434x902 | 3.9 GB | Mouse brain slice | 56 | 28 |
| X089-Mousebrain | 7 GB | Large mouse brain | 100 | 50 |

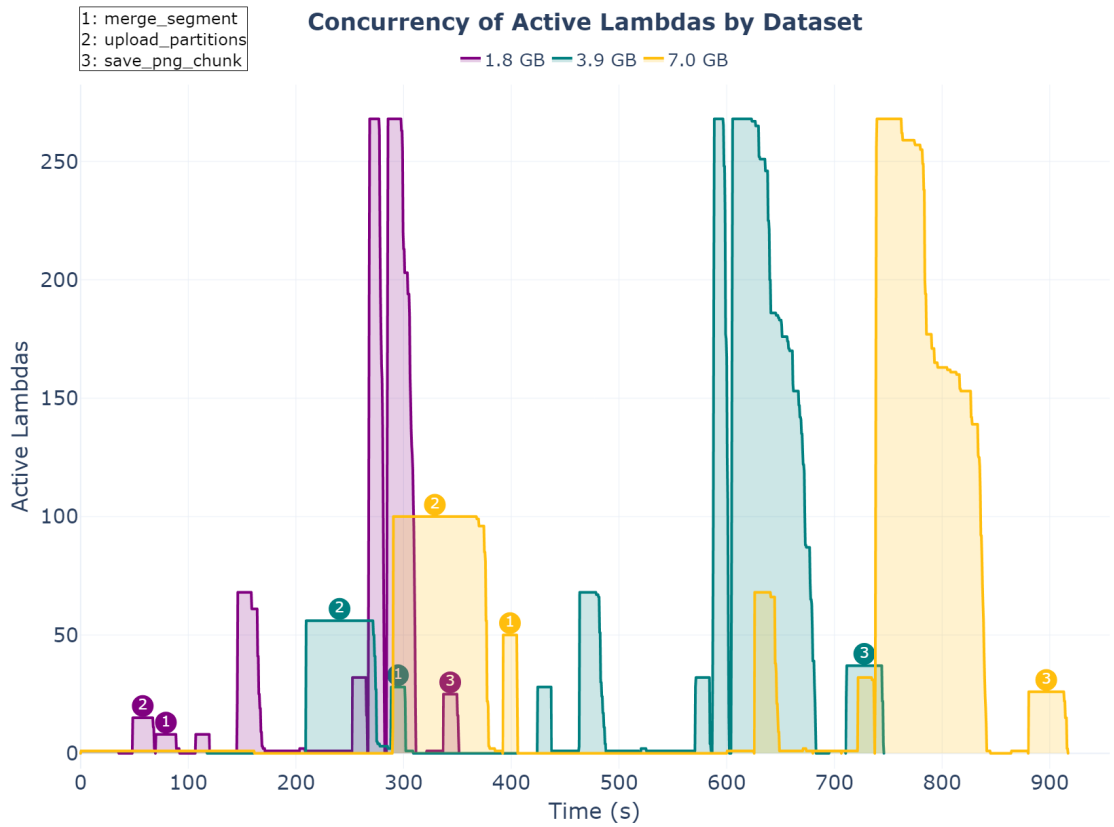
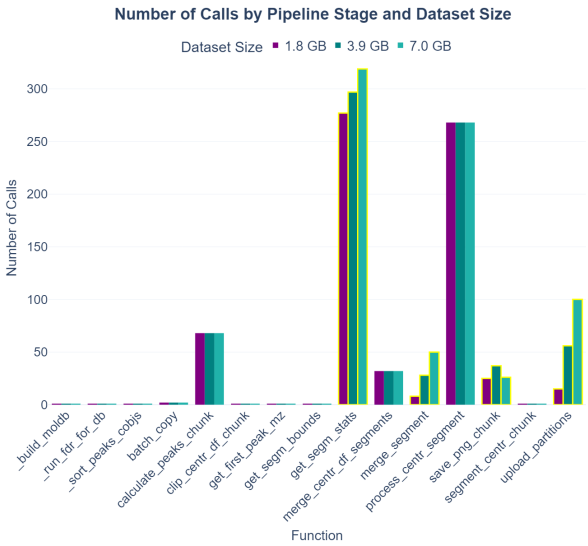


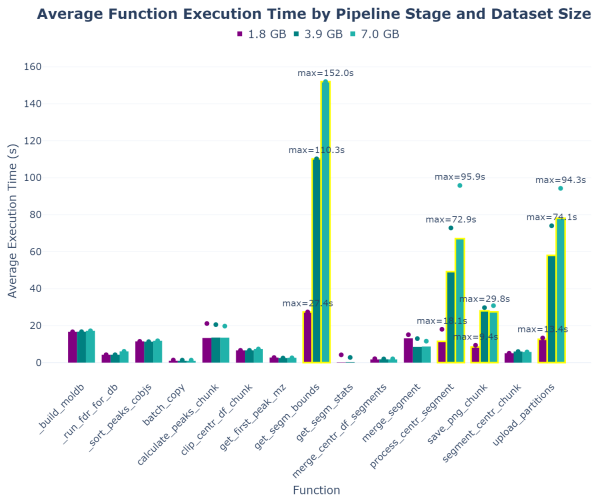

Figure 42: Concurrency of active AWS Lambda functions over time for three dataset sizes (1.8 GB, 3.9 GB, and 7.0 GB). The plot highlights the *merge_segment*, *upload_partitions*, and *save_png_chunk* functions to illustrate how their level of parallelism increases with dataset size.

In **Figure 42**, we analyze the concurrency patterns of Lambda functions when scaling the dataset size from 1.8 GB to 7 GB. As the dataset grows, we observe increases in total execution time and the overall concurrency across the pipeline stages. Dataset-dependent stages- `merge_segment`, `upload_partitions`- significantly scale their concurrency.

Examining **Figure 43**, the total number of Lambda invocations (**subfigure 43a**) notably rises for dataset-dependent stages, which correlates directly with dataset size. While the lightweight tasks in `get_segm_stats` see a significant increase in the number of calls, the heaviest compute stages—especially `process_centr_segment`—demonstrate marked increases in average execution times as datasets grow. This behavior is likely driven by amplified I/O traffic and data-fetch latency, as reflected in **Figure 44**, whereas `get_segm_bounds` is the stage with the most significant average exe-




(a) Total number of calls per pipeline stage for each dataset size.




(b) Average execution time per pipeline stage across dataset sizes.

Figure 43: Stacked comparison of function invocation volumes and execution latencies as dataset size increases.

cution time and shows the largest runtime increase as dataset size grows-peaking at 152s in the 7GB dataset-, since it runs once sequentially at the start of `load_ds_parallel`.

Data transfer volumes (**Fig. 44**) grow in tandem with dataset size, and reads consistently exceed writes. As in the database-size scaling in the previous experiment, the database-dependent `process_centr_segment` stage's write traffic also rises in proportion to dataset size. However, its read volume does not increase linearly: the smallest dataset presents more inbound traffic than the mid and large-size run.

Meanwhile, `upload_partitions` and `merge_segment` show a steady, proportional increase in both received and sent data across both runs as they implement the core of `load_ds_parallel`. By comparison, lighter stages such as `save_png_chunk` and `get_segm_stats` also see higher network traffic as the dataset grows, although their increase is much more modest.

Finally, **Figure 45b** shows that both CPU and memory usage significantly increase, especially when executing the 7GB dataset. The average CPU usage per Lambda function grows from approximately 7% for the 1.8GB dataset to around 30% for the 7GB dataset. Similarly, peak memory consumption rises considerably, from about 163MB to nearly 546MB. Ideally, a well-scalable system should maintain relatively stable resource utilization even with increasing workloads. Therefore, this upward trend indicates that some stages are not scaling as efficiently as intended, despite remaining fully functional.

These results, particularly the notable increase in peak memory consumption (from a median of 185MB to approximately 550MB), suggest that further scaling optimizations are possible and should be a focus for future work.

### 7.9.3 Optimizing Data Ingestion with DataCockpit

To determine a practical chunk size for distributed ingestion, we used DataCockpit on PyRun together with Dataplug. Dataplug provides lazy, cloud-aware partitioning of raw scientific data in object stores by building lightweight indexes and issuing byte-range reads on demand, which is especially important in metabolomics where formats like imzML/ibd are heterogeneous and cannot be split arbitrarily. In practice, we (1) refactored upload_partitions into a benchmarkable function, (2) let DataCockpit slice the dataset at a series of target chunk sizes (e.g., 30–130 MB), and (3) had

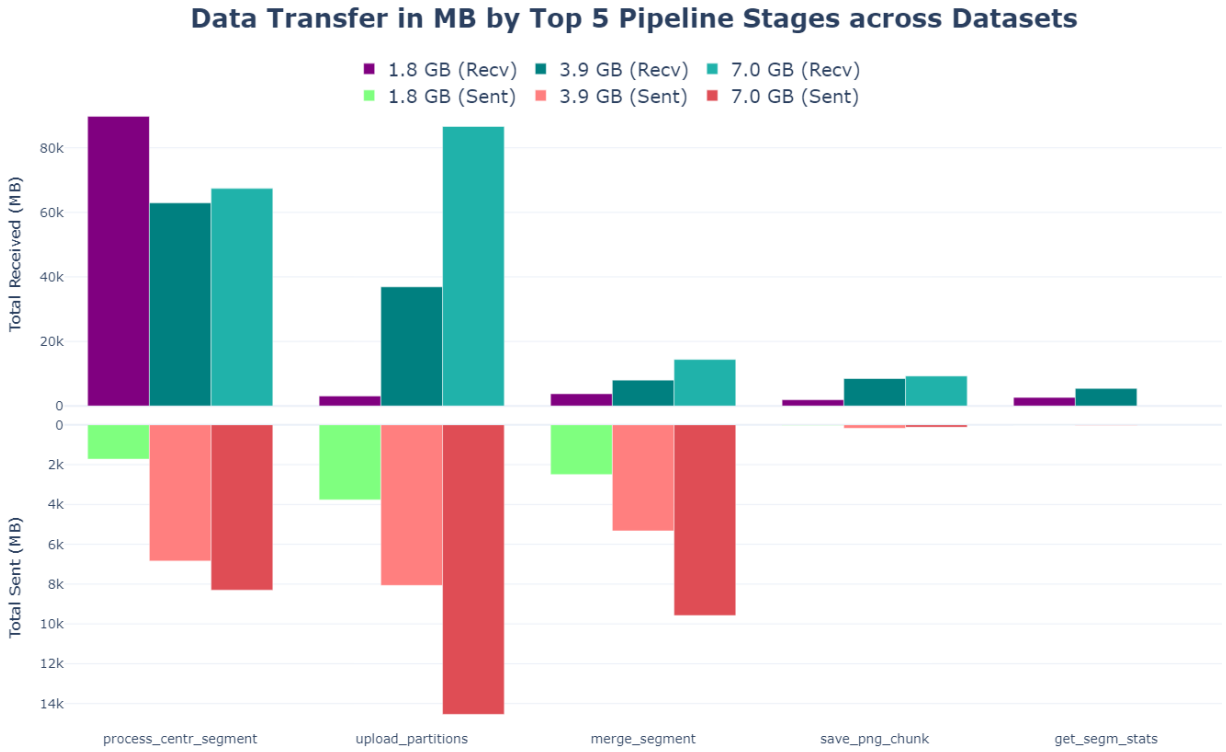
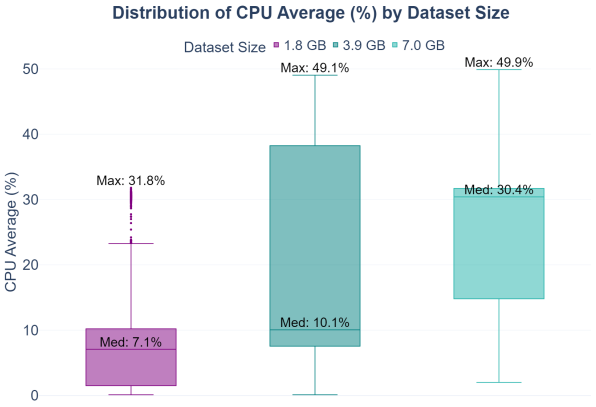
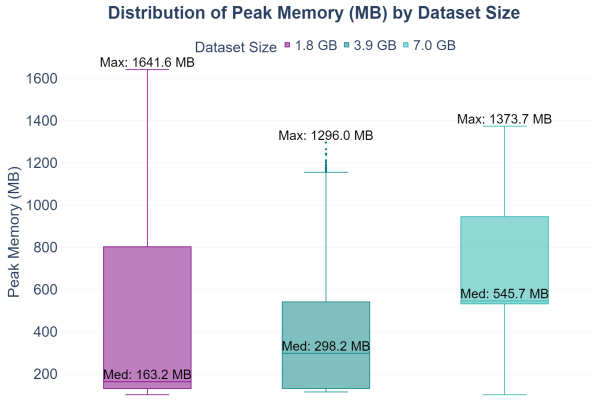



Figure 44: Data transfer volume (MB) for the five most I/O-heavy functions, showing both received (positive) and sent (negative) bytes across three dataset sizes.

DataCockpit execute our function over each slice to measure elapsed time. We then cross-validated the outcome by running the isolated stage inside the original pipeline to check that the average execution times and chunk sizes results match. This workflow avoids rewriting data, enables rapid,




(a) Distribution of average CPU utilization (%) by dataset size.




(b) Distribution of peak memory usage (MB) by dataset size.

Figure 45: Comparison of resource usage distributions across different dataset sizes: (a) average CPU utilization, (b) peak memory usage.

reproducible sweeps over partition sizes, and yields an empirical choice of chunk size for the ingestion stage.

As we can observe in the results presented in **Figure 47**, across all three dataset sizes (1.8 GB, 3.9 GB, 7.0 GB), 50 MB consistently minimized the stage's elapsed time in both the original pipeline and the DataCockpit version (see the two "Stage Execution Time by Chunk Size" panels: the curves attain their minima at 50 MB). Using 130 MB as the baseline (chosen to approximate prior practice that commonly adopts 128 MB block/chunk sizes in cloud analytics [96]), the measured speedups at 50 MB were 1.21–1.22× for 1.8 GB (17.3–18.2% faster), 1.14–1.16× for 3.9 GB (12.6–13.4% faster), and 1.10–1.13× for 7.0 GB (9.1–11.7% faster). Importantly, the original and DataCockpit runs show the same optimum (50 MB) and closely aligned effect sizes, confirming that the benchmarking pathway reflects pipeline behavior and is suitable for parameter tuning. We also observe that the total execution time of `upload_partitions` does not increase sharply with dataset size when the chunk size is well chosen (compare the per-dataset traces at 50 MB).

A 50 MB chunk size strikes a balance between cold-start/serialization overheads and function scheduling costs (which favor larger chunks) and I/O locality and per-Lambda memory (which favor smaller chunks). At 50 MB, tasks remain short and memory-moderate while reads/writes are large enough to amortize per-invocation overheads; this yields higher effective throughput without increasing peak memory per worker.

However, the speedup diminishes as dataset size grows. Moreover, ingestion contributes a smaller fraction of the end-to-end wall time and other stages (including sequential steps) dominate. Consequently, while ingestion tuning is worthwhile, the overall pipeline speedup is modest when measured end-to-end.

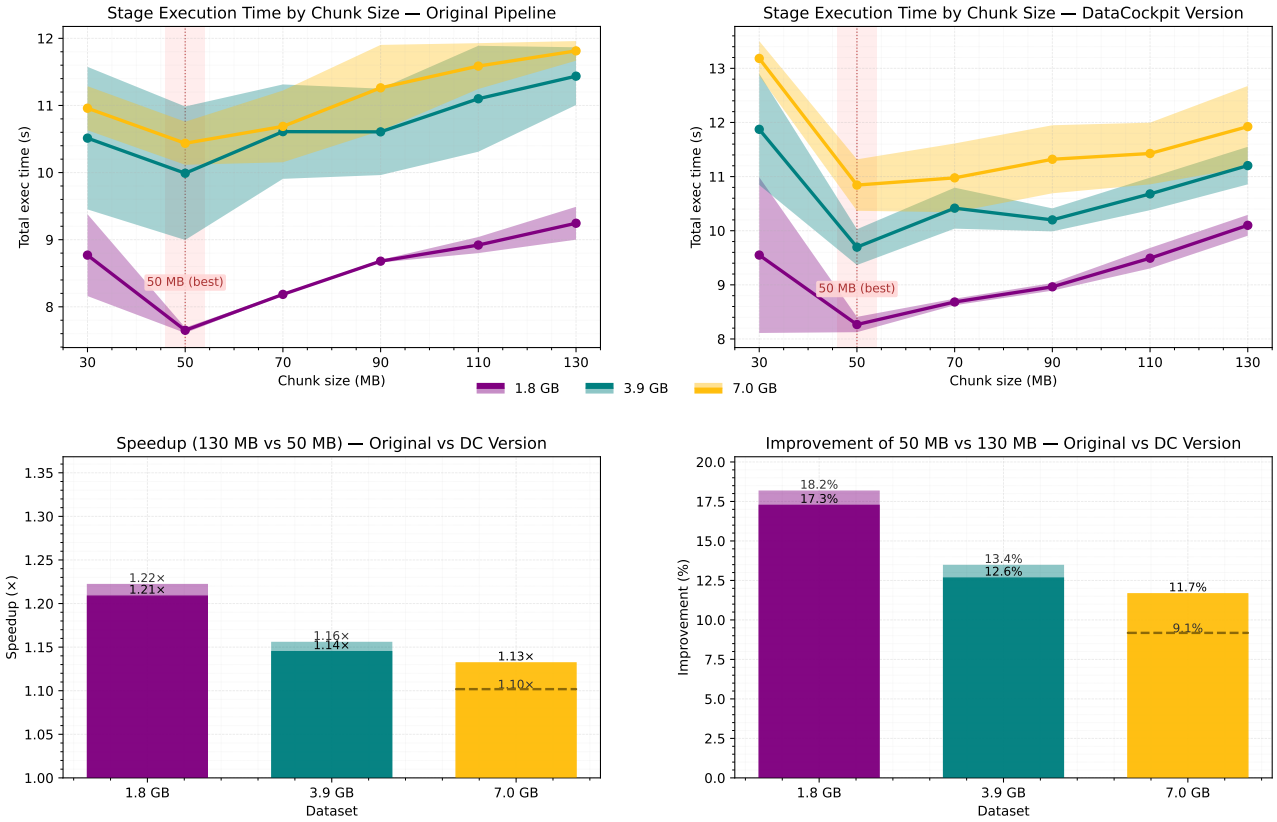



Figure 46: Ingestion benchmarking of upload_partitions with DataCockpit and pipeline validation

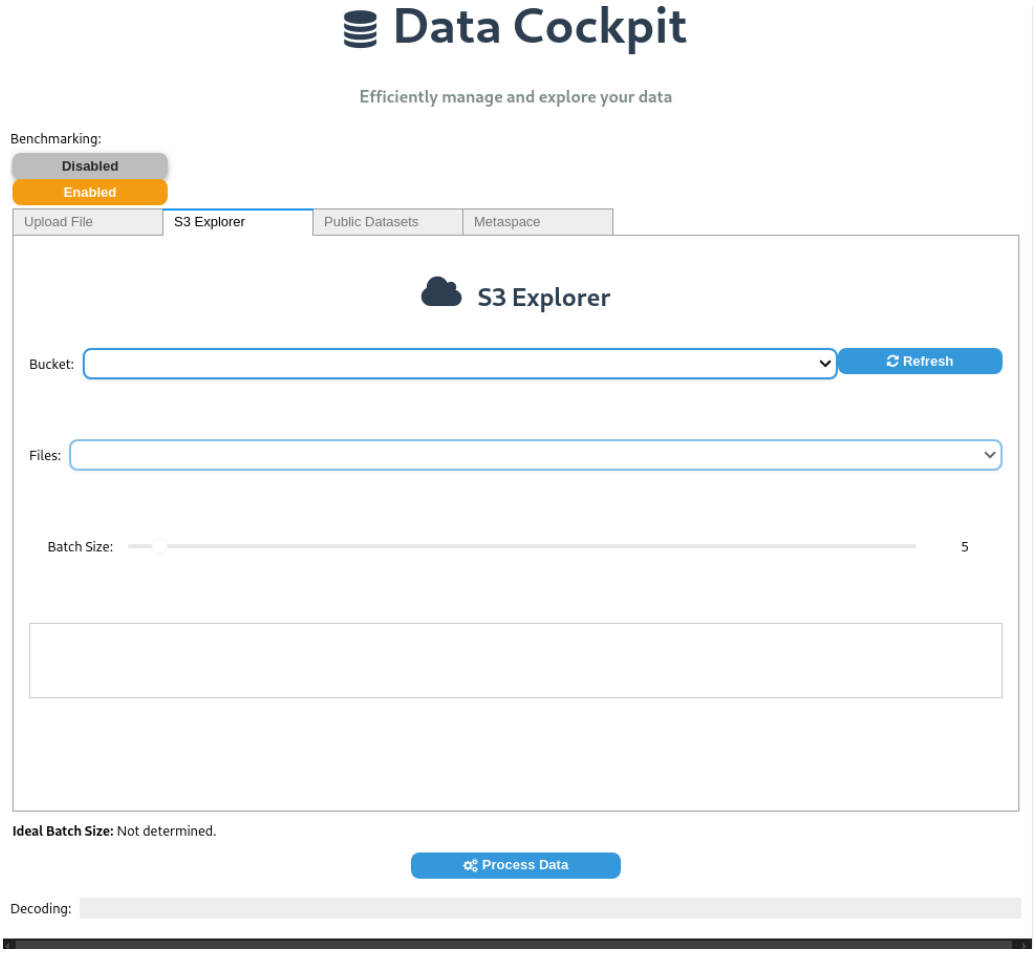



Figure 47: DataCockpit interface (S3 Explorer panel). This tool enables the user to manage and explore datasets (upload/S3/public/METASPACE), choose bucket and files, process your data with a selected batch size and, with benchmarking enabled, run a user function over a range of chunk sizes to find the optimal one.

### 7.9.4 Improving Pipeline Usability and Productivity with PyRun

As noted in previous sections, we rely on Pyrun to host the Lithops-METASPACE pipeline. Pyrun is a managed serverless Python platform that connects to the user's AWS account and abstracts packaging, provisioning, scaling, and monitoring. This "zero-setup" environment eliminates the need to build custom runtimes or configure cloud resources: users select a runtime (via environment.yml or a container), push code, and execute. All stages run as Lithops functions with object-storage I/O, while PyRun provides a single place to observe per-stage timelines, concurrency, and CPU/memory/disk/network metrics.

Moreover, PyRun lowers the barrier for non-expert users. Researchers with little to no background in programming or cloud operations can launch and monitor the full pipeline end-to-end. They interact with a simple execution interface, while PyRun and Lithops handle dependency resolution, autoscaling, retries, and fault isolation. This combination demonstrably increases usability and repeatability: complex, multi-stage cloud analytics become accessible, faster to iterate, and easier to share within teams.

### 7.9.5 Increasing Security of Pipeline Execution of Metaspace with SCONE

Metaspace has been selected to be ported to confidential computing in the Metabolomics use case. To do that, both Lithops and Metaspace (both developed in Python) had to be ported to SCONE. Lithops (verison 3.0.1 and later 3.5.0) had already been ported to SCONE, running on Python 3.10 and sup-

porting backends `localhost` and `k8s`. A new backend was introduced in Lithops 3.6.1: "*Singularity*"; it brings a new engine for working with "*threads*" instead of the traditional multiple OS processes spawned with the "*fork*" system call. Its full potential is achieved with `Python 3.13t`, that implements improvements on parallelization of threads. This new backend uses RabbitMQ to synchronize the division of work: 1) clients will submit tasks to a queue, i.e. "*publish*", and 2) Lithops will read from there, i.e. "*consume*", while 3) clients will wait until finished. Metaspace uses Lithops as an engine to distribute workload among backend nodes in the cloud, but it is exclusively compatible with Python 3.8; and this poses a challenge to the new approach of Lithops Singularity. Hence, in order to arrange the proper compatibility, URV forked Metaspace and managed to provide a compatible version with Python 3.13[10].

KIO Networks provided the appropriate hardware to perform the activities of research required to port the system. There is a Kubernetes cluster consisting of 4 nodes, being one control-plane (16 CPUs and 65GB RAM) e 3 workers (32 CPUs and 65GB RAM each). This infrastructure, supported the setup of 6 Lithops Singularity replicas, 1 MinIO server, and 1 RabbitMQ instance. Lithops Singularity + Metaspace and MinIO have been *sconified*, i.e., ported to SCONE. RabbitMQ has not been ported yet; it is developed in the Erlang platform, which Scontain has only experimental support. Other confidential computing protection mechanisms are used: attestation, via LAS (Local Attestation Service) and CAS (Configuration and Attestation Service); file shield, that encrypts data on disk; and network shield, that enforces network communication over TLS between attested applications.

The pipeline was setup with the chosen dataset related to the *brain* of a *Rattus norvegicus (rat)*[11] and has 705MB – very small in comparison with other datasets. Firstly, the native Metaspace was executed to learn the general behavior, to than proceed to the sconified in simulation, to later execute in full confidential computing mode, using the TEE. The table [26] shows the durations of the native and the sconified simulation and hardware modes executions in minutes. The values were calculated based on the duration converging towards the center, with quartile 2 and 3 as lower bound and upper bound respectively. The charts [48] show the duration of the selected executions done with the dataset in the three modes, plus a comparison, sorted ascending by duration.

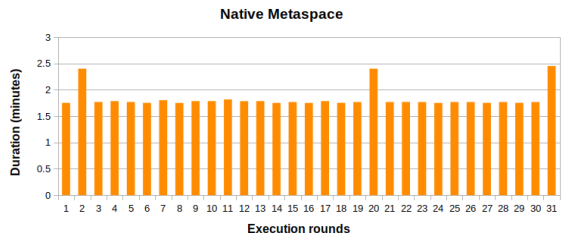
| Version | Maximum | Minimum | Mean | Quartile 2 | Quartile 3 | Orders of Magnitude |
|---|---|---|---|---|---|---|
| native | 1.784 | 1.768 | 1.774 | 1.767 | 1.785 | 1 x |
| sconified (simulation) | 10.132 | 9.982 | 10.046 | 9.978 | 10.135 | 5.662 x more |
| sconified (hardware) | 90.839 | 25.030 | 44.963 | 25.017 | 91.230 | 25.34 x more |

Table 26: Metaspace pipeline – duration comparison between native and sconified versions (in minutes).

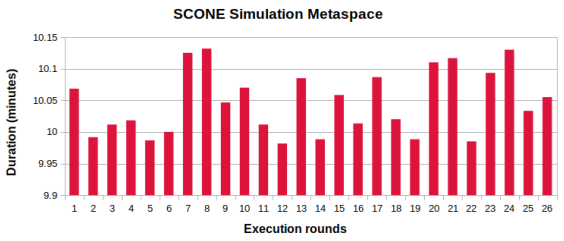
The baseline for the analysis is the native mode execution, with an average duration of 1.774 minutes for complete and successful executions. Adjusting the backend for Lithops Singularity on sconified simulation mode, the duration mean value increased to 10.046 minutes for complete and successful executions, representing $\approx 5.662$ orders of magnitude to native.

When moving to hardware, we observed very high slowness and consequent aborted executions. Hence, many other adjustments had to be made on server side: increase the EPC (Enclave Page Cache) from the initial **1GB** to **16GB**; increase RAM on each of the 3 worker nodes to **256GB**, and later increasing to **320GB** RAM; reduce from 6 to 3 Lithops Singularity instances, in order to configure them with nearly the total amount of memory in each node. However, no successful execution in sconified hardware mode could be concluded. The increasing duration times observed in [48c] and [48d] refer to actions troubleshooting to find the point of failure. It refers to the calling of the function "`merge_centr_df_segments()`", when there are **32 simultaneous** activations.

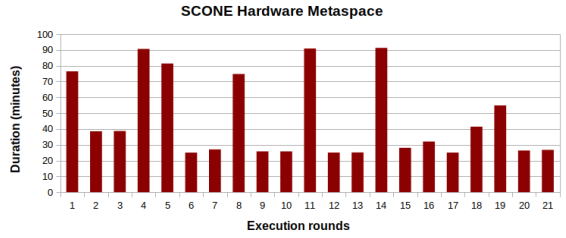
[10] METASPACE annotation pipeline adaptation to python 3.13 – https://github.com/GEizaguirre/metaspace-py13/
[11] AstraZeneca//Xenograft https://metaspace2020.org/dataset/2016-09-21_16h06m53s



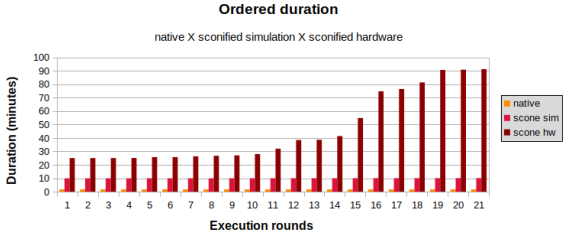

(a) Native.




(b) SCONE Simulation.




(c) SCONE Hardware.




(d) Comparing the executions.

Figure 48: Cloud-Edge Continuum Workshop 2024 posters session.

Every program executed in confidential computing builds up an "**enclave**", with the total amount of RAM necessary reserved to it from the startup time, that summarizes the whole TCB (Trusted Computing Base), and each enclave is an OS process. In the case of Metaspace using the Lithops Singularity as engine, the RAM reserved at startup is **8GB** of HEAP memory and it is the minimum for having it running. When Metaspace divides the work and distributes the activities, it is not using "*threads*" in the new Lithops Singularity backend adequately; it still uses the "`fork()`" system call mechanism, that will spawn an OS process on the server side. Moreover, despite dividing the work among the worker nodes, each enclave forked had a second "`fork()`" executed, and this would eventually exhaust the RAM of the server and abort the whole pipeline execution; that is, there is at least 16GB RAM per double-forked enclave, summed up to the amount of processes Metaspace assessed to be possible to push, plus 8GB of the root process. Configuration modifications were made in Metaspace parameters, like "`worker_processes`", "`max_workers`", "`runtime_timeout`"; and also in the Kubernetes "`resources » limits » cpu`" settings of resources, but nothing changed Metaspace behavior; this indicates that choosing memory and number of activities is related to inner business logic.

It is not clear whether this double-forking behavior comes from the Python interpreter or is related to the "*futures*" nature of Lithops division and distribution of work; from the SCONE side, only the translation of system calls issued by the programs are made and sent to the CPU. Regardless of the reason for this, the main lesson is that Metaspace, *as-is*, does not benefit from the Lithops Singularity that uses the "*threads*" mechanism. Encryption calculation, program contexts motion to-and-from CPU and enclave add a very heavy layer of computation that affects the overall performance of an application, i.e. there is a trade-off of performance for security; and in confidential computing, as a rule, often more resources made available can get the good job done – please find other details in deliverable "**D4.2 Data Broker Reference Implementation**". However, in the metabolomics use case, more specifically, with Metaspace, a large amount of hardware has already been put at service of the application when ported to SCONE, but the specificity of its internal mechanisms indicate that more hardware will always be necessary to complete a pipeline. An "endless" hardware resources scenario has a prohibitive cost; therefore, the way out here is to promote a better integration of Metaspace with Lithops Singularity (*threads* mechanism) to adequately be supported by SCONE without the need to

`fork()` enclaves – each copy of HEAP reserved to the enclave takes a significant time; whereas with threads, this would not be necessary. Lithops Singularity, on the other hand, has been successfully ported to SCONE and can benefit from the confidential computing protection mechanisms.

### 7.10 Use-case repositories

The following list specifies the source-code locations for the pipeline in this use case experiment:

- METASPACE main repository: `https://github.com/metaspace2020`
- METASPACE pipeline: `https://github.com/metaspace2020/metaspace`
- Lithops-METASPACE: `https://github.com/metaspace2020/Lithops-METASPACE`
- DataPlug: `https://github.com/CLOUDLAB-URV/dataplug/tree/master`
- DataCockpit: `https://github.com/ubenabdelkrim/data-cockpit`
- OffSample: `https://github.com/metaspace2020/offsample`
- METASPACE fork to python 3.13: `https://github.com/GEizaguirre/metaspace-py13/`
- Confidential Lithops and Metaspace: `https://github.com/neardata-eu/scone-artifacts/`

### 7.11 Discussion and final remarks

This use case shows that, since the stages differ substantially in their computational and I/O behavior, a fully serverless implementation is preferable to a fixed cluster for this pipeline. When we vary the database size, the system scales up parallelism in database-dependent stages (e.g., centroid matching/processing); per-invocation CPU and memory stay broadly stable, indicating that additional load is handled by more workers, not larger ones. When we vary the dataset size, higher parallelism appears mainly in dataset-dependent stages (ingestion and segment merging). In both experiments, a few sequential steps—notably `get_segm_bounds` and parts of database preparation—impose a visible floor on total stage execution time, consistent with Amdahl's law.

On the ETL side, moving from the single-Lambda `load_ds` to the distributed `load_ds_ parallel` enables larger datasets and exposes configurable parameters (`ds_n_chunks`, `ds_n_segms`) to control concurrency. Using DataCockpit and Dataplug to sweep chunk sizes and validating against the isolated stage, we identified 50 MB as the optimal chunk size for `upload_ partitions`. The measured speedups over a 130 MB baseline were 1.21–1.22× (1.8 GB), 1.14–1.16× (3.9 GB), and 1.10–1.13× (7.0 GB), and the optimal point is consistent between the benchmarking function provided to DataCockpit and the pipeline. Importantly, the absolute time of `upload_ partitions` grows slowly with dataset size, indicating that we are balancing cold starts/serialization with I/O locality and per-Lambda memory.

On the other hand, there are some limitations we must take into account. As datasets grow, the relative speedup of ingestion decreases. Besides, ingestion contributes a smaller share of the end-to-end time while other stages—especially the sequential ones—dominate. At 7 GB, some stages show higher average CPU and peak memory per invocation, which suggests that their resource usage increases with input size.

Regarding usability, hosting Lithops-METASPACE on PyRun provides a managed entry point (runtime packaging, basic provisioning, and monitoring). This reduced setup effort and helped us iterate on chunk sizes and collect per-stage telemetry without managing cluster infrastructure. PyRun is an abstraction layer rather than a performance component; its value here was mainly to simplify execution and measurement for both expert and non-expert users.

Finally, this experiment also demonstrates the importance of a federated Data Space: by unifying access to heterogeneous metabolomics repositories and enabling on-the-fly partitioning through DataCockpit and DataPlug, we were able to seamlessly connect large-scale data access with scalable serverless execution. This integration highlights how standardized access and dynamic partitioning are enablers for reproducible, high-performance metabolomics pipelines.

### 7.12 Future work

Future work should focus on resource provisioning and on reducing the time dominated by single-function stages. This is necessary because, at larger inputs, some stages show rising per-invocation CPU and memory, indicating suboptimal sizing. Next steps include reviewing per-stage memory/CPU settings to limit this growth and keep resource use stable as datasets increase. These actions are likely to deliver larger end-to-end improvements than further ingestion tuning alone.

Moreover, the heterogeneity and constraints of data providers must be addressed. Although our dedicated connectors mitigate fragmentation by integrating Metaspace, AWS Open Registry, MetaboLights, and Metabolomics Workbench into DataCockpit and DATOMA, these providers continue to expose different protocols and often require API keys with usage policies that impose restrictions such as rate limits, bandwidth caps, or selective access conditions. Future efforts should therefore explore:

- Standardization of API contracts to simplify integration across repositories.
- Token and credential management mechanisms that allow pipelines to handle API key constraints transparently.
- Caching and replication strategies to reduce repeated API calls and alleviate rate-limit restrictions.
- Adaptive orchestration that can reconfigure pipelines dynamically based on provider-imposed quotas or access rules.

Addressing these challenges will be essential for realizing the full potential of a federated, FAIR-compliant Data Space that consistently supports large-scale, serverless metabolomics pipelines.

# 8 Conclusions

## 8.1 Challenges addressed and achieved results

The use cases are the core of the NearData project, as they provide state-of-the-art, relevant examples of current challenges in the life sciences. Each of the use cases was selected carefully in order to be representative of such a complex background. This is the reason why the connectors developed in each of them are publicly delivered so they can contribute to the community, and there is no need to repeat this work, but simply to adapt the connectors to each specific constraint the community may have. We provide in the following list the different challenges that have been resolved for each use case.

**Use case Genomics Epistasis**

- **Machine learning**: current techniques were not adapted to the millions of features of the human genome and its millions of variants. Thus, a strategy was devised to be able to apply a combinatorial strategy aligning such statistical constraints to state-of-the-art machine learning techniques.
- **Data ingestion**: not only is the amount of variants extreme, but the volume of the data sets is as well, and it poses a bottleneck for the use case. For this, a two-fold strategy was devised: on one hand, parallelizing such ingestion on small partitions that are read at once, on the other hand, shuffling the contents of such partitions among them to avoid internal structures of the datasets.
- **Orchestration**: bringing the serverless paradigm to the HPC to make it user-friendly comes with the challenge of deciding how many resources to allocate to it. The larger the number, the longer the wait time to run, and hence degrades the time to service. In this use case, we have devised an auto-scaling mechanism to make a better resource allocation strategy considering current demands in the system.
- **Analysis**: finally, analysing the results of the machine learning model was not trivial and was hugely impacted by the hyperparameter selection. Such selection is complex as there are many parameters that, combined with the complexity of the data challenges, make it difficult to find the most relevant and its best setting. In this work, we have worked on novel strategies for hyperparameter selection that have led to a better identification of variants leading to disease.

**Use case Computer-Assisted Video Surgery**

- **Flexible content-based video stream indexing**: although the video stream is immutable, its contents can be indexed in multiple ways. Those indices determine the models that can be applied and their usage. Thus, we need to allow flexible re-indexing based on the end-user target. StreamSense is a framework that has been built in the context of this project and allows low-latency video indexing.
- **Scalable semantic search**: dealing with tiered video streams for long retention periods requires managing large amounts of index data. In our use-cases this can turn into very large querying costs in terms of memory and size. So indexing techniques must account for those trade-offs. For this challenge, Vector DBs were integrated, which, combined with the StreamSense framework, addressed the challenge.
- **Programmatic search interface**: semantic search algorithms enable user-friendliness in our use case. However, training for extreme-data scenarios is not straightforward and requires exploration of the best techniques.
- **Secure Federated Learning with Flower and SCONE**: In this use case, sensitive surgical video data remains on-site at each hospital while federated learning, orchestrated by the Flower

framework, enables collaborative AI model training. SCONE ensures that all computations and model updates occur within Trusted Execution Environments (TEEs), guaranteeing the confidentiality and integrity of both data and models, even against privileged attackers. Model updates are encrypted and integrity-checked during transmission, preventing data leakage or tampering throughout the collaborative process.

**Use Case Transcriptomics**

- **Resource-intensive application**: STAR aligner requires an indexed human genome to be loaded into the worker node's memory in order to perform alignment on a FASTQ file. By using the newest release of the human genome from Ensembl [85], we were able to reduce the requirement from 80GB to 35GB, therefore saving resources, reducing costs, and improving processing time significantly (12x faster). Moreover by using early stopping feature we skip processing of FASTQ files that will end up with mapping rate below our threshold using intermediate results/metrics. With this optimizations we can increase the alignment throughput by 19.5% (alignment is the most resource intensive step).
- **Large-scale dataset of RNA-sequences**: For the Transcriptomics Atlas use case, we need to process about 130TB of FASTQ data. With the optimizations (Tab. 19) implemented, this is very efficient both in term of time and cost. We reduced computation costs with spot instnaces, faster/smaller STAR index, early stopping (and other optimizations) and this allows possible extension of the Atlas with additional tissues which will make it more comprehensive and useful.

**Use Case Genomics**

- **Managing redundant and accessible genomic data across stream and object interfaces**: the genomics research community is exploring streaming data infrastructures to support both real-time stream processing and long-term object storage. Enabling efficient dual-mode access - where the same genomic data must be accessible both as a stream and as a compressed object - introduces significant complexity. A key challenge is enabling such dual-access through a unified API. The challenge was addressed via integrating stream-native data connectors and serverless analytics frameworks, allowing dual-mode access, compression, and scalable processing.
- **Reducing complexity in deploying scalable genomic stream analytics**: executing genomic analytics requires deploying and managing complex data processing engines. Such engines come with substantial operational overheads, becoming a barrier in scenarios where rapid deployment and elasticity are crucial. The challenge lies in finding scalable, low-footprint alternatives that can deliver performance without the burden of managing heavyweight infrastructure. Through the implementation of the Nexus FASTQGzip streamlet, we offer such efficient ingestion and analytics without compromising performance.

**Use Case Metabolomics**

- **Serverless, on-demand execution**: The metabolomics pipeline mixes CPU/memory-intensive database stages with I/O-bound dataset stages and a few sequential steps. Because of the heterogeneity of each stage, a fixed cluster would either sit idle in light stages or fall short during bursts. We therefore used a serverless, stage-oriented design so concurrency is granted only where needed, avoiding resource blocking and improving the balance between cost and performance.
- **Data ingestion bottleneck and scaling limits**: We replaced the original load_ds stage, which ran the whole imzML/ibd ingest in a single function and limited the dataset size, with a distributed sort-style loader that estimates m/z bounds, maps windowed reads, and reduced per-segment outputs. This parallelizes ingestion and enables larger datasets without inflating per-task memory.

- **Partitioning trade-offs in serverless ingestion**: Choosing how many chunks to use when partitioning the dataset is a tuning problem in our distributed loader: smaller chunks cut per-task memory and seek time but raise overheads, whereas larger chunks do the opposite. Using DataCockpit (with Dataplug), we benchmarked the upload_partitions logic across chunk sizes and selected 50 MB as the time-optimal setting versus a 130 MB baseline, a result that also held when validating the stage in isolation.

- **Data Space heterogeneity and provider constraints**: Datasets arrive from multiple providers with different APIs, protocols, and limits (bandwidth, access policies). We built connectors and integrated them into DataCockpit/DATOMA to standardize access and support partitioned reads across sources. This reduces fragmentation, though full interoperability and guaranteed openness remain open issues.

- **Secure pipeline computation with Metaspace + Lithops + SCONE**: It has been observed a great opportunity to tighten the integration between Metaspace and Lithops using this new Singularity backend; with that put in place, Metaspace could benefit in full of the confidential computing features SCONE has available.

### 8.2 Future work

In the genomics epistasis use case, the future steps are two-fold: on one hand, improving the Lithops-HPC auto-scaling ecosystem with better resource management and parallelization techniques. On the other hand, work on the analysis of the obtained data and results and biologically validating the results with clinical trials, which, due to the nature of such trials, take several years to carry out.

For the surgery use case, one line goes towards exploring the enhancements of the Amazon AWS service, now allowing for S3 vectors. Combined with the NEARDATA results, leveraging Vector DBs might prove a powerful tool. On the other hand, AI is an essential element in the use case. As such, examining how to implement federated learning within the German's Surgical AI Hub is a promising research line to be explored to translate the research advancements into clinical practice.

Next steps for the transcriptomic use case include expanding the Atlas dataset with additional tissues. Following this extension, the data will be processed through the established pipeline and collect its results, which will then be prepared, analyzed, and published on open-source data repositories such as Zenodo.

For the Genomics use case, after the integration of NEARDATA technologies into the pipeline and seeing its potential, further research will be made on how to integrate such technologies into the public health infrastructures. One direction of it is the deployment of real-time outbreak monitoring systems leveraging stream-based ingestion and analytics to enable faster detection and response.

Regarding the metabolomics use case, the next steps will center on tuning resource provisioning (per-stage memory/CPU and latency control) to keep resource usage stable as datasets scale, and strengthening data-space integration by standardizing APIs, handling credentials and quotas transparently, and adding caching/replication and adaptive orchestration. Together, these efforts aim to deliver larger end-to-end gains than ingestion tuning alone while making the pipeline more robust across diverse data providers.

### 8.3 List of data connectors

In the following list, we briefly summarize each of the connectors developed in the project:

- **Data partitioning**: this connector partitions the dataset into small chunks so the read can be parallelized.

- **Data shuffle**: the connector merges with the partitioning one and not only partitions but shuffles the content of each of the partitions to break inherent structures of VCF data.

- **Auto-scaling connector**: this connector interacts with HPC and Lithops-HPC to dynamically adjust the resources assigned to the compute backend.

- **ML accelerator connector**: this connector is built inside Lithops-HPC to be able to manage GPUs as well as CPUs currently. It can be expanded to other devices with minimal effort.
- **HPC connector**: this connector facilitates the connection of Lithops with HPC systems, delivering what we call Lithops-HPC architecture.
- **Indexer controller connector**: this connector deploys AI models developed by NCT to generate embeddings from video frames and replies to semantic queries.
- **Indexed genome connector**: this connector allows performing STAR alignments on each worker. This is achieved through an NFS-based approach, as it performed better than alternatives using object storage or embedding indexes into virtual machines.
- **Dataplug SRA format reader**: this connector enables the Dataplug architecture to recognize SRA to FASTQ formats. This connector allows seamless partitioning and distribution of chunks of FASTQ file, while the data on the storage remains in SRA format.
- **Nexus FASTQGzip streamlet**: this connector performs GZip data compression on subsets of DNA sequences based on an input stream of data. Each of the partition start offsets is stored as an object stage to perform parallel access efficiently.
- **FaaStream Streaming K-Pop job**: this connector ingests a parallel stream of FASTQ data from Pravega at the FASTQ record level. Then the records are read and compute hashes for every subsequence of length $k$ within each genomic sequence, allowing for shuffling and fast aggregation of FASTQ-based data.
- **AWS Open Registry, Metaspace, MetaboLights, and Metabolomics Workbench connectors**: those connectors eliminate the need for users to work directly with heterogeneous APIs, offering a unified access point to metabolomics data. Combined with Dataplug, they transform diverse datasets into partitioned analysis-ready blocks suitable for parallel and reproducible processing.

### 8.4 Summary of KPIs achieved

Table 27 summarizes the KPIs achieved by each of the use cases.

### 8.5 Available repositories

Table 28 contains the list of repositories where use cases or connectors are made available.

| KPI | Use Case | Summary of results |
|---|---|---|
| KPI-1 | Epistasis(GWD) | Lithops-HPC connector improves data ingestion in GWD pipeline by **36x**. |
| KPI-1 | Epistasis(GWD) | Hyperparameter selection improves performance by **5x**. |
| KPI-5 | Epistasis(GWD/MDR) | Cyclomatic-Complexity reveals **1.5x** fewer execution paths. |
| KPI-5 | Epistasis(GWD/MDR) | Yaqin's metrics reveals **1.6x** fewer branches, loops and nesting depth. |
| KPI-3 | Epistasis(GWD/MDR) | The auto-scaler improves execution time of MDR use-case by **1.5x**. |
| KPI-2 | Surgery | StreamSense achieve low video frame indexing latency between **63ms** and **360ms**. |
| KPI-2 | Surgery | Semantic video search latency is around **30**ms for large collections of surgical video. |
| KPI-1 | Surgery | Data transfer savings in AI loading. Integrating with PyTorch data transfers are reduced **between 83.79% and 99.83%**. |
| KPI-4 | Surgery | Enhanced encryption of files with TEE, access control meachisnms. The security of the system was rigorously validated through adversarial testing and TEE attestation, confirming that both the confidentiality and integrity of model updates and training data were consistently enforced. |
| KPI-1 | Transcriptomics | Early stopping technique have increased alignment throughput by **19.5%**. |
| KPI-1 | Transcriptomics | Use of a newer release of human genome index has resulted in execution times improvements of up to **12x** and smaller STAR index file (from 85GB to 30GB). |
| KPI-1 | Transcriptomics | The usage of spot instances reduced, on average, **50%** of the execution cost. |
| KPI-1 | Genomics | The Nexus FASTQGzip streamlet provides a good compression vs speed trade-off. It achieves **3.8x** better compression ratio than plain FASTQ and **13.2x** faster data processing than using FASTQ Gzip files. |
| KPI-1 | Genomics | FaaStream is up to **65.14%** cheaper than Flink running the genomics `Kpop` job. |
| KPI-1 | Metabolomics | Depending on the size, we get a speed-up on processing time ranging from **1.13x to 1.22x** faster. |
| KPI-4 | Metabolomics | Achieved full confidential computing support (data at rest, in transit and in use) on cloud storage service (MinIO) and FaaS Lithops Singularity, aided by SCONE mechanisms and using the TEE; achieved partial confidential computing support (data at rest and in transit) due to limitations of the ported system (Metaspace is not fully compliant with Lithops Singularity + SCONE). |

Table 27: Summary of KPIs achieved on the use cases

| Description | URL |
|---|---|
| GWD Lithops-HPC | https://gitlab.bsc.es/datacentric-computing/lithops-hpc-genomics |
| MDR MPI | https://gitlab.bsc.es/datacentric-computing/mpi-genomics-mdr |
| MDR Lithops-HPC | https://github.com/neardata-eu/lithops-hpc/tree/main/examples/mdr |
| Lithops-HPC | https://github.com/neardata-eu/lithops-hpc |
| Lithops-HPC user guide | https://gitlab.bsc.es/datacentric-computing/lithops-hpc-examples |
| Data shuffle connector | https://github.com/danielBCN/dataplug/tree/filesystem |
| Auto-scaling connector | https://gitlab.bsc.es/datacentric-computing/lithops_telemetry_forecasting |
| Surgery-FL training models | https://github.com/KirchnerMax/FL-EndoViT |
| Surgery-FL use case | https://gitlab.com/nct_tso_public/challenges/miccai2024/FedSurg24 |
| Surgery-FL tools | https://gitlab.com/nct_tso_public/challenges/miccai2024/snippet |
| Transcriptomics Atlas | https://github.com/neardata-eu/transcriptomics-atlas-sano |
| Salmon pipeline | https://github.com/Kamilbur/salmonless |
| Nexus | https://github.com/neardata-eu/nexus |
| FaaStream | https://github.com/neardata-eu/faastream |
| METASPACE | https://github.com/metaspace2020 |
| Lithops-METASPACE | https://github.com/metaspace2020/Lithops-METASPACE |
| DataPlug | https://github.com/CLOUDLAB-URV/dataplug/ |
| DataCockpit | https://github.com/ubenabdelkrim/data-cockpit |
| OffSample | https://github.com/metaspace2020/offsample |

Table 28: Repositories with data connectors, use cases, and related tools

## References

[1] Prometheus Authors 2014-2025, "Prometheus," 2025.

[2] Prometheus Authors 2014-2025, "When to use the pushgateway," 2025.

[3] MetricFire Blogger, "Prometheus pushgateways - everything you need to know," Oct 12, 2023.

[4] Prometheus Developers, "Prometheus pushgateway github," 2025. GitHub repository.

[5] Prometheus Authors 2014-2025, "Prometheus alert manager," 2025.

[6] Prometheus Developers, "Prometheus alertmanager github," 2025. GitHub repository.

[7] The Apache Software Foundation, "Apache airflow," 2025.

[8] A. B. Arevalo, D. C. Tejeda, A. Call, P. G. López, and R. N. Castell, "Enhancing hpc with serverless computing: Lithops on marenostrum5," in 2024 IEEE 32nd International Conference on Network Protocols (ICNP), pp. 1–6, 2024.

[9] M.-A. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann, "Gekkofs - a temporary distributed file system for hpc applications," in 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 319–324, 2018.

[10] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 26, no. 1, pp. 43–49, 1978.

[11] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in Proceedings of the 18th International Conference on Data Engineering (ICDE), pp. 673–684, IEEE, 2002.

[12] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), pp. 792–803, VLDB Endowment, 2004.

[13] A. Stefan, V. Athitsos, and G. Das, "The move-split-merge metric for time series," IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 6, pp. 1425–1438, 2013.

[14] G. Box and G. M. Jenkins, Time Series Analysis: Forecasting and Control. Holden-Day, 1976.

[15] Y.-W. Cheung and K. S. Lai, "Lag order and critical values of the augmented dickey–fuller test," Journal of Business & Economic Statistics, vol. 13, no. 3, pp. 277–280, 1995.

[16] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?," Journal of Econometrics, vol. 54, no. 1-3, pp. 159–178, 1992.

[17] B. Iglewicz and D. C. Hoaglin, How to Detect and Handle Outliers. The ASQC Basic References in Quality Control: Statistical Techniques, Milwaukee, WI: ASQC Quality Press, 1993.

[18] "National center for tumor diseases (germany)." `https://www.nct-dresden.de/en/homepage`, 2025.

[19] L. Maier-Hein et al., "Surgical data science for next-generation interventions," Nature Biomedical Engineering, vol. 1, no. 9, pp. 691–696, 2017.

[20] L. Maier-Hein, M. Eisenmann, D. Sarikaya, K. März, T. Collins, A. Malpani, J. Fallert, H. Feussner, S. Giannarou, P. Mascagni, H. Nakawala, A. Park, C. Pugh, D. Stoyanov, S. S. Vedula, K. Cleary, G. Fichtinger, G. Forestier, B. Gibaud, T. Grantcharov, M. Hashizume, D. Heckmann-Nötzel, H. G. Kenngott, R. Kikinis, L. Mündermann, N. Navab, S. Onogur, T. Roß, R. Sznitman, R. H. Taylor, M. D. Tizabi, M. Wagner, G. D. Hager, T. Neumuth, N. Padoy, J. Collins, I. Gockel, J. Goedeke, D. A. Hashimoto, L. Joyeux, K. Lam, D. R. Leff, A. Madani, H. J. Marcus, O. Meireles, A. Seitel, D. Teber, F. Ückert, B. P. Müller-Stich, P. Jannin, and S. Speidel, "Surgical data science – from concepts toward clinical translation," vol. 76, p. 102306.

[21] "Amazon kinesis video streams examples." `https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/examples.html`, 2024.

[22] "Pravega." `https://cncf.pravega.io`, 2023.

[23] "Pravega - gstreamer connector." `https://github.com/pravega/gstreamer-pravega`, 2024.

[24] "Guidance for semantic video search on aws." `https://aws.amazon.com/es/solutions/guidance/semantic-video-search-on-aws/`, 2024.

[25] "Azure ai video indexer." `https://azure.microsoft.com/en-us/products/ai-video-indexer`, 2024.

[26] D. Ouyang, B. He, A. Ghorbani, N. Yuan, J. Ebinger, C. P. Langlotz, P. A. Heidenreich, R. A. Harrington, D. H. Liang, E. A. Ashley, et al., "Video-based ai for beat-to-beat assessment of cardiac function," Nature, vol. 580, no. 7802, pp. 252–256, 2020.

[27] R. Gracia-Tinedo, F. Junqueira, T. Kaitchuck, and S. Joshi, "Pravega: A tiered storage system for data streams," in Proceedings of the 24th International Middleware Conference, Middleware '23, (New York, NY, USA), p. 165–177, Association for Computing Machinery, 2023.

[28] "Apache pulsar - overview of tiered storage." `https://pulsar.apache.org/docs/2.11.x/tiered-storage-overview`, 2023.

[29] "Amazon kinesis." `https://aws.amazon.com/es/kinesis`, 2023.

[30] "Dell streaming data platform." `https://www.dell.com/en-us/dt/storage/streaming-data-platform.htm`, 2023.

[31] Milvus, "Product faq." `https://milvus.io/docs/product_faq.md#What-is-the-maximum-dataset-size-Milvus-can-handle`, 2024.

[32] Weaviate, "Vector indexing." `https://weaviate.io/developers/weaviate/concepts/vector-index#vector-cache-considerations`, 2024.

[33] F. P. Junqueira, I. Kelly, and B. Reed, "Durability with bookkeeper," ACM SIGOPS operating systems review, vol. 47, no. 1, pp. 9–15, 2013.

[34] R. Gracia-Tinedo, J. Sampé, E. Zamora, M. Sánchez-Artigas, P. García-López, Y. Moatti, and E. Rom, "Crystal: Software-defined storage for multi-tenant object stores," in USENIX FAST'17, pp. 243–256, 2017.

[35] K. Echihabi, K. Zoumpatianos, and T. Palpanas, "Scalable machine learning on high-dimensional vectors: From data series to deep network embeddings," in WIMS'20, pp. 1–6, 2020.

[36] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, K. Yu, Y. Yuan, Y. Zou, J. Long, Y. Cai, Z. Li, Z. Zhang, Y. Mo, J. Gu, R. Jiang, Y. Wei, and C. Xie, "Milvus: A purpose-built vector data management system," in Proceedings of the 2021 International Conference on Management of Data, SIGMOD '21, (New York, NY, USA), p. 2614–2627, Association for Computing Machinery, 2021.

[37] J. Yuan, H. Wang, L. Xiao, W. Zheng, J. Li, F. Lin, and B. Zhang, "A formal study of shot boundary detection," IEEE Transactions on Circuits and Systems for Video Technology, vol. 17, no. 2, pp. 168–186, 2007.

[38] A. F. Smeaton, P. Over, and A. R. Doherty, "Video shot boundary detection: Seven years of trecvid activity," Computer Vision and Image Understanding, vol. 114, no. 4, pp. 411–418, 2010.

[39] R. Guo, X. Luan, L. Xiang, X. Yan, X. Yi, J. Luo, Q. Cheng, W. Xu, J. Luo, F. Liu, et al., "Manu: a cloud native vector database management system," Proceedings of the VLDB Endowment, vol. 15, no. 12, pp. 3548–3561, 2022.

[40] "Streamsense code repository." `https://github.com/neardata-eu/video-stream-indexing`, 2024.

[41] H. Alhajj, M. Lamard, P.-h. Conze, B. Cochener, and G. Quellec, "Cataracts," 2021.

[42] C. I. Nwoye et al., "Rendezvous: Attention mechanisms for the recognition of surgical action triplets in endoscopic videos," Medical Image Analysis, vol. 78, p. 102433, 2022.

[43] Z. Wang, B. Lu, Y. Long, F. Zhong, T.-H. Cheung, Q. Dou, and Y. Liu, "Autolaparo: A new dataset of integrated multi-tasks for image-guided surgical automation in laparoscopic hysterectomy," 2022.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[45] C.-J. Yu et al., "Lightweight deep neural networks for cholelithiasis and cholecystitis detection by point-of-care ultrasound," Computer Methods and Programs in Biomedicine, vol. 211, p. 106382, 08 2021.

[46] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One model to learn them all," 2017.

[47] "Vectordbbench: A benchmark tool for vectordb." `https://github.com/zilliztech/VectorDBBench`, 2024.

[48] "General data protection regulation (GDPR) – legal text."

[49] O. f. C. Rights (OCR), "Health information privacy." Last Modified: 2025-06-27T11:38:47-0400.

[50] F. R. Kolbinger, M. Kirchner, K. Pfeiffer, S. Bodenstedt, A. C. Jenke, J. Barthel, M. R. Carstens, K. Dehlke, S. Dietz, S. Emmanouilidis, G. Fitze, L. Leitermann, S. T. Mees, S. Pistorius, C. Prudlo, A. Seiberth, J. Schultz, K. Thiel, D. Ziehn, S. Speidel, J. Weitz, J. N. Kather, M. Distler, and O. L. Saldanha, "Appendix300: A multi-institutional laparoscopic appendectomy video dataset for computational modeling tasks." ISSN: 3067-2007 Pages: 2025.09.05.25335174.

[51] C. A. Gomes, T. A. Nunes, J. M. Fonseca Chebli, C. S. Junior, and C. C. Gomes, "Laparoscopy grading system of acute appendicitis: new insight for future trials," vol. 22, no. 5, pp. 463–466.

[52] M. Carstens, F. M. Rinner, S. Bodenstedt, A. C. Jenke, J. Weitz, M. Distler, S. Speidel, and F. R. Kolbinger, "The dresden surgical anatomy dataset for abdominal organ segmentation in surgical data science," vol. 10, no. 1, p. 3. Publisher: Nature Publishing Group.

[53] L. Maier-Hein, M. Wagner, T. Ross, A. Reinke, S. Bodenstedt, P. M. Full, H. Hempe, D. Mindroc-Filimon, P. Scholz, T. N. Tran, P. Bruno, A. Kisilenko, B. Müller, T. Davitashvili, M. Capek, M. Tizabi, M. Eisenmann, T. J. Adler, J. Gröhl, M. Schellenberg, S. Seidlitz, T. Y. E. Lai, B. Pekdemir, V. Roethlingshoefer, F. Both, S. Bittel, M. Mengler, L. Mündermann, M. Apitz, A. Kopp-Schneider, S. Speidel, H. G. Kenngott, and B. P. Müller-Stich, "Heidelberg colorectal data set for surgical data science in the sensor operating room."

[54] M. Wagner, B.-P. Müller-Stich, A. Kisilenko, D. Tran, P. Heger, L. Mündermann, D. M. Lubotsky, B. Müller, T. Davitashvili, M. Capek, et al., "Comparative validation of machine learning algorithms for surgical workflow and skill analysis with the heichole benchmark," Medical Image Analysis, vol. 86, p. 102770, 2023.

[55] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. d. Mathelin, and N. Padoy, "EndoNet: A deep architecture for recognition tasks on laparoscopic videos."

[56] M. Pfeiffer, I. Funke, M. R. Robu, S. Bodenstedt, L. Strenger, S. Engelhardt, T. Roß, M. J. Clarkson, K. Gurusamy, B. R. Davidson, L. Maier-Hein, C. Riediger, T. Welsch, J. Weitz, and S. Speidel, "Generating large labeled data sets for laparoscopic image processing tasks using unpaired image-to-image translation."

[57] A. Leibetseder, S. Kletz, K. Schoeffmann, S. Keckstein, and J. Keckstein, "GLENDA: Gynecologic laparoscopy endometriosis dataset," in MultiMedia Modeling (Y. M. Ro, W.-H. Cheng, J. Kim, W.-T. Chu, P. Cui, J.-W. Choi, M.-C. Hu, and W. De Neve, eds.), pp. 439–450, Springer International Publishing.

[58] K. Schoeffmann, H. Husslein, S. Kletz, S. Petscharnig, B. Muenzer, and C. Beecks, "Video retrieval in laparoscopic video recordings with dynamic content descriptors," vol. 77, no. 13, pp. 16813–16832.

[59] J. Yoon, J. Lee, S. Heo, H. Yu, J. Lim, C. Song, S. Hong, S. Hong, B. Park, S. Park, W. J. Hyung, and M.-K. Choi, "hSDB-instrument: Instrument localization database for laparoscopic and robotic surgeries," in Medical Image Computing and Computer Assisted Intervention – MICCAI 2021 (M. de Bruijne, P. C. Cattin, S. Cotin, N. Padoy, S. Speidel, Y. Zheng, and C. Essert, eds.), pp. 393–402, Springer International Publishing.

[60] A. Leibetseder, S. Petscharnig, M. J. Primus, S. Kletz, B. Münzer, K. Schoeffmann, and J. Keckstein, "Lapgyn4: a dataset for 4 automatic content analysis problems in the domain of laparoscopic gynecology," in Proceedings of the 9th ACM Multimedia Systems Conference, MMSys '18, pp. 357–362, Association for Computing Machinery.

[61] V. S. Bawa, G. Singh, F. KapingA, I. Skarga-Bandurova, A. Leporini, C. Landolfo, A. Stabile, F. Setti, R. Muradore, E. Oleari, and F. Cuzzolin, "ESAD: Endoscopic surgeon action detection dataset."

[62] N. Valderrama, P. R. Puentes, I. Hernández, N. Ayobi, M. Verlyk, J. Santander, J. Caicedo, N. Fernández, and P. Arbeláez, "Towards holistic surgical scene understanding," vol. 13437, pp. 442–452.

[63] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "Vivit: A video vision transformer," in Proceedings of the IEEE/CVF international conference on computer vision, pp. 6836–6846, 2021.

[64] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, pp. 1273–1282, PMLR. ISSN: 2640-3498.

[65] D. Batić, F. Holm, E. Özsoy, T. Czempiel, and N. Navab, "EndoViT: pretraining vision transformers on a large collection of endoscopic images," vol. 19, no. 6, pp. 1085–1091.

[66] D. Caldarola, B. Caputo, and M. Ciccone, "Improving generalization in federated learning by seeking flat minima," in Computer Vision – ECCV 2022 (S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, eds.), pp. 654–672, Springer Nature Switzerland.

[67] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015. arXiv:1512.03385 [cs].

[68] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature Verification using a "Siamese" Time Delay Neural Network," in Advances in Neural Information Processing Systems, vol. 6, Morgan-Kaufmann, 1993.

[69] Z. Qu, X. Li, R. Duan, Y. Liu, B. Tang, and Z. Lu, "Generalized federated learning via sharpness aware minimization."

[70] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. d. Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework."

[71] T. Li, M. Sanjabi, A. Beirami, and V. Smith, "Fair resource allocation in federated learning."

[72] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in Proceedings of the 35th International Conference on Machine Learning, pp. 5650–5659, PMLR. ISSN: 2640-3498.

[73] A. Casamassimi, A. Federico, M. Rienzo, S. Esposito, and A. Ciccodicola, "Transcriptome profiling in human diseases: new advances and perspectives," International journal of molecular sciences, vol. 18, no. 8, p. 1652, 2017.

[74] "Salmon repository," 2023.

[75] "STAR repository," 2023.

[76] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras, "Star: ultrafast universal rna-seq aligner," Bioinformatics, vol. 29, no. 1, pp. 15–21, 2013.

[77] "DESeq2," 2023.

[78] "NCBI NIH: The Sequence Read Archive." https://www.ncbi.nlm.nih.gov/sra/docs, 2024.

[79] "Sequence Read Archive Data Working Group." https://dpcpsi.nih.gov/council/sradwg, 2024.

[80] "NIH's Sequence Read Archive, the world's largest genome sequence repository: Openly accessible on AWS." https://aws.amazon.com/blogs/industries/nihs-sequence-read-archive-the-worlds-largest-genome-sequence-repository-openly-accessible-on-aws/, 2024.

[81] M. Oh and L. Zhang, "Deepmicro: deep representation learning for disease prediction based on microbiome data," Scientific Reports, vol. 10, no. 1, p. 6026, 2020.

[82] E. Pasolli, D. T. Truong, F. Malik, L. Waldron, and N. Segata, "Machine learning meta-analysis of large metagenomic datasets: tools and biological insights," PLOS Computational Biology, vol. 12, no. 7, p. e1004977, 2016.

[83] J. Meizner, P. Kica, S. Lichołai, and M. Malawski, "Analysis of index access pattern in STAR rna-seq aligner," in Proceedings of the Seventeenth ACC Cyfronet AGH HPC Users' Conference (KU KDM 2025), pp. 31–32, 2025.

[84] P. Kica, S. Lichołai, M. Orzechowski, and M. Malawski, "Optimizing Star Aligner for High Throughput Computing in the Cloud," in 2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops), pp. 162–163, IEEE, 2024.

[85] F. J. Martin, M. R. Amode, A. Aneja, O. Austine-Orimoloye, A. Azov, I. Barnes, A. Becker, R. Bennett, A. Berry, J. Bhai, S. Bhurji, A. Bignell, S. Boddu, P. R. Branco Lins, L. Brooks, S. B. Ramaraju, M. Charkhchi, A. Cockburn, L. Da Rin Fiorretto, C. Davidson, K. Dodiya, S. Donaldson, B. El Houdaigui, T. El Naboulsi, R. Fatima, C. G. Giron, T. Genez, G. S. Ghattaoraya, J. G. Martinez, C. Guijarro, M. Hardy, Z. Hollis, T. Hourlier, T. Hunt, M. Kay, V. Kaykala, T. Le, D. Lemos, D. Marques-Coelho, J. C. Marugán, G. Merino, L. Mirabueno, A. Mushtaq, S. Hossain, D. N. Ogeh, M. P. Sakthivel, A. Parker, M. Perry, I. Piližota, I. Prosovetskaia, J. G. Pérez-Silva, A. Salam, N. Saraiva-Agostinho, H. Schuilenburg, D. Sheppard, S. Sinha, B. Sipos, W. Stark, E. Steed, R. Sukumaran, D. Sumathipala, M.-M. Suner, L. Surapaneni, K. Sutinen, M. Szpak, F. Tricomi, D. Urbina-Gómez, A. Veidenberg, T. Walsh, B. Walts, E. Wass, N. Willhoft, J. Allen, J. Alvarez-Jarreta, M. Chakiachvili, B. Flint, S. Giorgetti, L. Haggerty, G. Ilsley, J. Loveland, B. Moore, J. Mudge, J. Tate, D. Thybert, S. Trevanion, A. Winterbottom, A. Frankish, S. E. Hunt, M. Ruffier, F. Cunningham, S. Dyer, R. Finn, K. Howe, P. W. Harrison, A. D. Yates, and P. Flicek, "Ensembl 2023," Nucleic Acids Research, vol. 51, pp. D933–D941, 11 2022.

[86] X. Didelot and P. Ribeca, "KPop: accurate and scalable comparative analysis of microbial genomes by sequence embeddings," Genome Biology, vol. 26, p. 170, June 2025.

[87] F. Versaci, L. Pireddu, and G. Zanetti, "Kafka interfaces for composable streaming genomics pipelines," in IEEE EMBS International Conference on Biomedical & Health Informatics (BHI'18), pp. 259–262, 2018.

[88] R. Gracia-Tinedo, F. Junqueira, and T. Kaitchuck, ""Back to the byte": Towards byte-oriented semantics for streaming storage," in ACM/IFIP Middleware'24 (Industrial Track), pp. 43–49, 2024.

[89] A. Arjona, P. García-López, and D. Barcelona-Pons, "Dataplug: Unlocking extreme data analytics with on-the-fly dynamic partitioning of unstructured data," in IEEE CCGrid'24, pp. 567–576, 2024.

[90] "Documentation for memory configuration in aws lambda." `https://docs.aws.amazon.com/lambda/latest/dg/configuration-memory.html`, 2024.

[91] "Kpop repository." `https://github.com/PaoloRibeca/KPop`, 2025.

[92] "Kpop nextflow workflows repository." `https://github.com/ryanmorrison22/kpop-workflow`, 2025.

[93] "Neardata project - nexus." `https://github.com/neardata-eu/nexus`, 2025.

[94] "Neardata project - faastream." `https://github.com/neardata-eu/faastream`, 2025.

[95] K. Ovchinnikova, V. Kovalev, L. Stuart, and T. Alexandrov, "Offsampleai: artificial intelligence approach to recognize off-sample mass spectrometry images," BMC Bioinformatics, vol. 21, no. 1, p. 129, 2020.

[96] H. Bian, T. Sha, and A. Ailamaki, "Using cloud functions as accelerator for elastic data analytics," Proc. ACM Manag. Data, vol. 1, June 2023.