NEARDATA

Funded by the European Union

HORIZON EUROPE FRAMEWORK PROGRAMME

NEARDATA

(grant agreement No 101092644)

Extreme Near-Data Processing Platform

D5.1 First release of KPI benchmarks in all use cases and data connector libraries

Due date of deliverable: 30-04-2024
Actual submission date: 30-04-2024

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	76
WP/Task related to this document	WP5 / T5.2,T5.3,T5.4,T5.5,T5.6
WP/Task responsible	BSC-CNS
Leader	Aaron Call (BSC)
Technical Manager	Xavier Roca (URV)
Quality Manager	Raúl Gracia (DELL)
Author(s)	Aaron Call (BSC), Lorena Alonso (BSC), Gonzalo Gómez (BSC), Xavier Roca (URV), Raúl Gracia (DELL), Sean Ahearne (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), Theodore Alexandrov (EMBL)
Partner(s) Contributing	BSC, URV, DELL, UKHS, EMBL, SCO, NCT
Document ID	NEARDATA_D5.1_Public.pdf
Abstract	This deliverable will describe the first validation benchmarks in the five use cases. First release of open data connectors.
Keywords	Use cases, Data connectors, Data throughput, Genomics, Metabolomics, Transcriptomics, Lithops, Pravega, SCONE

History of changes

Version	Date	Author	Summary of changes
0.1	11-03-2023	Aaron Call (BSC), Sebastian Bodenstedt (NCT), Max Kirchner (NCT)	Initial structure in place.
0.2	25-03-2023	Aaron Call (BSC), Lorena Alonso (BSC), Gonzalo Gómez (BSC), Xavier Roca (URV), Raúl Gracia (DELL), Sean Ahearne (DELL), Ger Hallissey (DELL), Sebastian Bodenstedt (NCT), Max Kirchner (NCT), Reuben Docea (NCT), Paolo Ribeca (UKHS), Andre Miguel (SCO), Maciej Malawski (Sano), Piotr Kica (Sano), Jan Przybyszewski (Sano), Sabina Lichołai (Sano), Kamil Burkiewicz (Sano), Theodore Alexandrov (EMBL)	First Draft.
1.0	30-04-2024	Aaron Call (BSC)	Final version.

Table of Contents

List of Abbreviations and Acronyms

API	Application Programming Interface
CC	Creative Commons
CSV	Comma-separated values
CV	Cross-Validation
DOI	Digital Object Identifier
ETL	Extract Transform Load
GWD	Genome-Wide Discovery
HPC	High-Performance Computing
KPI	Key Performance Indicator
MDR	Multi Dimensionality Reduction
MN4	MareNostrum 4 Supercomputer
MN5	MareNostrum 5 Supercomputer
MPI	Message Passing Interface
T2D	Type 2 Diabetes
AUC	Area Under the Receiver Operating Characteristic Curve
CGS	Computer-guided Surgery
CV	Cross-Validation
DoA	Description of the Action
DSAD	Dresden Surgical Anatomy Dataset
EKS	Elastic Kubernetes Service
ENA	European Nucleotide Archive
ExAC	Exome Aggregation Consortion
FaaS	Function as a Service
FL	Federated Learning
FLOPS	Floating Point Operations per Second
Gnom AD	Genome Aggregation Database
GPFS	General Parallel File System
HPC	High Performance Computing
INSDC	International Nucleotide Sequence Database Collaboration
LD	Linkage Disequilibrium
ML	Machine Learning

NVMe	Non-Volatile Memory Express
PRACE	Partnership for Advanced Computing in Europe
RES	Red Española de Supercomputación
ROC	Receiver Operating Characteristic
TCGA	The Cancer Genome Atlas

1 Executive summary

Deliverable 5.1 comprises the first release of KPI benchmarks on all use cases and data connector libraries. This deliverable covers the work performed during the first 14 months of the project.

In this deliverable, we introduce extreme data use cases and the data connectors implemented towards making them more efficient in terms of the KPIs defined in the Description of the Action (DoA). To do so we present a set of experiments designed to evaluate the targeted KPIs and the results obtained so far. All of the use cases have already met some of the targeted objectives (in terms of KPI) while others are in progress. Moreover, we describe how some connectors allow to integration of NEARDATA architecture elements such as Lithops, SCONE, or Pravega and allow for the use cases to be more efficient in terms of data throughput and/or transfer, as well as to reduce data latency and allow secure data transfers in the cloud continuum.

For each of the use cases we describe:

- The reasons why the use cases utilized are extreme data with quantitative metrics.
- The datasets that will be used to benchmark the use cases' performance.
- The set of data connectors developed and integrated.
- The benchmarks used to evaluate the use cases as well as the KPIs achieved on each benchmark. We particularly evaluate how our data connectors improve efficiency in terms of data transfer reductions as well as improved data merging and sorting when applicable.
- For those use cases that have been integrated with NEARDATA architecture components, we describe how the integration has been made and its benefits.

Finally, 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 Introduction

The NEARDATA project is rooted in the near-data processing paradigm, which can be defined as the strategy of moving computational resources close to the data, instead of transferring the data to the processor. NEARDATA aims to establish a robust near-data infrastructure that acts as a mediator for data flows between object storage and data analytics platforms throughout the compute continuum.

The movement of resources close to the data is mainly achieved by what we call data connectors. A data connector is defined as an element capable of communicating computational resources with a variety of data. This data can be structured or unstructured, which brings a heterogeneity challenge commonly found in the omics sciences. The connections are made in such a way that maximizes the efficiency of each use case on an ad-hoc basis according to the needs.

The data connectors help achieve NEARDATA's goals and will be demonstrated by our five omics use cases. Such use cases play a key role in the project. The selected use cases show an extreme data problem, making them a good fit to be improved using data connectors.

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.

3 Variants-Interactions Use-case

Current state of the art devoted their efforts to determining how single genome variants could be associated with complex diseases. However recent studies have shown that while a specific couple of variants may not be associated with the disease, the combination of both of them within a single genome may be. Thus, it is necessary to devote our efforts to studying the impact of variant interactions with such diseases. In this section, we present a use case for the identification of such interactions with two different but complimentary methodologies, which we define as separate use cases and experiments.

3.1 Why is the use case extreme data?

The discovery of variant groups associated with the disease involves the analysis of over 15 million variants. Studying all the possible combinations of pairs of variants represents more than $6x10^{13}$ tests. Using the Mare Nostrum 4 (MN4) architecture (165,888 CPU - 1.880 GB/core and 208 GPU - 16GB/each), this process is estimated to take approximately 9 days. This exemplifies how this use case represents an extreme data problem. On the other hand, the combinatorial analysis based on machine learning approaches requires the execution of over 112 million tests, which represents 3 days of continuous analysis using the whole MN4[1] architecture. Moreover, this calculus is done by computing only pairs of variants. However, the ideal scenario would be computing all the combinations possible with the 23 chromosomes. Thus, sets of 23 rather than just pairs. Consequently, the problem would take months of computations using the whole supercomputer.

3.2 Assessed Datasets

We work with three main datasets, two of them mostly utilized for the GWD use-case, and a third generated a synthetic datasheet to work on the MDR use-case.

- 70K dataset (Bonàs-Guarch et al., 2018)
  To address the variant interaction problem and assess the reliability of the models we used a pilot dataset, the 70KforT2D, a dataset that contains the genomic information for 70,127 individuals (12,931 diabetic, 57,196 non-diabetic) in 15,131,345 variants. Particularly, we evaluated the viability of MDR in a subset of this dataset which consists of 1,128 individuals, 601 non-diabetic and 527 diabetic, and the summary information of 1,883,192 genomic pairs. The pilot GWD model was assessed using a subset that consists of 105,896 variants and 22,802 individuals (11,401 diabetic and 11,401 non-diabetic).

- UKBioBank (Sudlow et al., 2015)
  To ensure the quality of the GWD model, its scalability, and the robustness of the predictor, we will use the UK Biobank, a genetic biobank that contains the genomic, clinical and environmental information for 422,000 European ancestry individuals (31,344 diabetic, 233,285 non-diabetic). Importantly, we have the genetic information for these individuals in 15,586,493 variants. This dataset will be used to accomplish two tasks: 1) validate the model trained in the 70KforT2D and 2) train the final model.

- Synthetic dataset
  To ensure the scalability of the MDR model, we will use a synthetic dataset that replicates the features of the aforementioned datasets. This dataset allows us to work on development environments so we can experiment with different improvements on the use case, as well as to analyze the behavior of the use case for different dataset sizes. This wouldn't be possible with a real dataset due to confidentiality constraints. Those constraints imply we can only access the dataset in the MareNostrum IV supercomputer, which is security-constrained due to being a production environment. On the other hand, only a few selected people are allowed access to the dataset itself. Thus, not all researchers and engineers may access it despite being granted access to the supercomputer. Moreover, the manipulation of those datasets is complex as we could easily invalidate the contained data. Consequently, a synthetic dataset becomes

optimal. This dataset is generated by replicating the features of the aforementioned, however, instead of being provided with real data, this data is fake. Although fake, the random data is generated taking into consideration that it could be found in a patient. With this, we can apply the workload over this dataset and extrapolate the results into the ones with data coming from true patients.

3.3 Description of the use-case

Complex diseases, such as Type 2 Diabetes (T2D), are caused by the simultaneous effect of multiple genomic variants and other environmental factors. During the last decades, the genomic study of T2D has been broadly approached with diverse methods. However, although hundreds of genomic variants are expected to contribute to disease development, the study of the relation between variants and T2D has been only analyzed in a single independent manner. There are different reasons to avoid this type of analysis. Among them, the analysis of variant interactions and their contribution to developing the disease is an extreme computational problem. Indeed, here we present two use cases where we tackle this type of analysis with different methods:

- Genome-Wide discovery (GWD): use machine learning methods to find groups of variants that, simultaneously, are associated with T2D.
- Multi Dimensionality Reduction (MDR): use statistical methods to discover pairs of variants which, synergically, contribute to the development of T2D.

GWD use case is depicted in figure 1 where it shows how the dataset containing the variations to be analysed. This dataset is utilized to train a machine learning model based on XGBoost. The optimal hyparameter adjustment estimation for the learning rate, split and number of trees, is utilized to 1) divide each partition in a training and test set, 2) train a model and generate a genomics predictor using the training set, and 3) test the output model in the test set. The results obtained will be used, in a combinational manner, as an input to the model recursively until finding a stable solution.

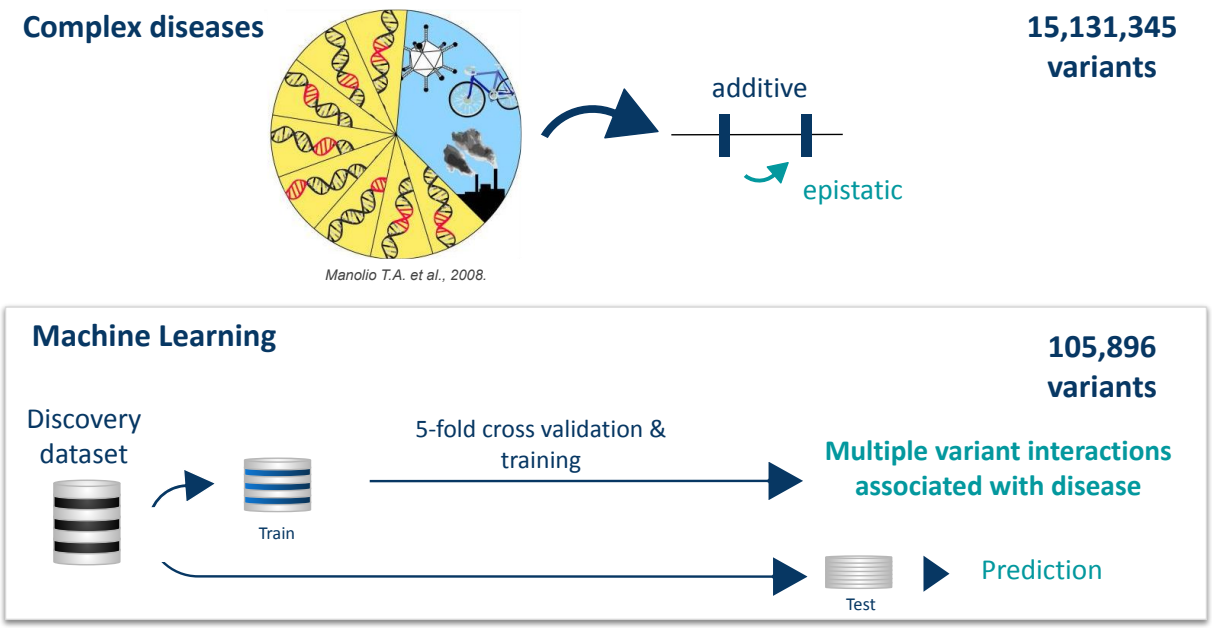


Figure 1: Genome-Wide discovery (GWD) use case overview

MDR use case is depicted in figure 2. The steps depicted are as follows:

- The first step consists of a k-fold cross-validation and is performed to avoid potential over-fitting. The dataset is subdivided into k parts where k-1 data parts are used for training and one part is used for testing. Since we generate k different distributions of the dataset, the following steps are going to be performed k times, one for each distribution.
- In the second step, for each variant pair we cross-tabulate genotypes for cases and controls, only using the individuals from the training dataset. This generates a 2 factors table (variant-variant), each one with 3 classes corresponding to the variant genotypes (AA, Aa, aa), ending with a 9-cell table with two dimensions: the number of cases and the number of controls.
- In step three, each of the tables is going to be transformed into a 1-dimension space using the case-control ratio from the training distribu- tion set as a threshold, T. The cells where the ratio of cases to controls is greater than T are going to be classified as 'high risk' and, otherwise, as 'low risk'. As a result of this step, we reduce the dimension of the problem from 2 classes to 1.
- In step four, each of the variant-variant tables is used to classify the individuals of the training set. The classification is done as follows: for each individual, we extract the variant-variant genotypes and check to which class of the multifactor table belongs. Then, if the class corresponds to a high-risk cell, we classify the individual as a case. Otherwise, we classify it as a control. After classifying every individual, we compare the predicted classes with the original labels, obtaining the miss-classification error of the variant-variant table.
- In step five, the model with the best miss-classification error is selected and the prediction power of the model is estimated using the indepen- dent test data as the percentage of patients correctly classified. After these steps are repeated for each of the possible cross-validation sets, the best and most consistent variant-variant combinations are selected. This means picking the ones that appear most times in the cross-validation sets as a top predictor pair.

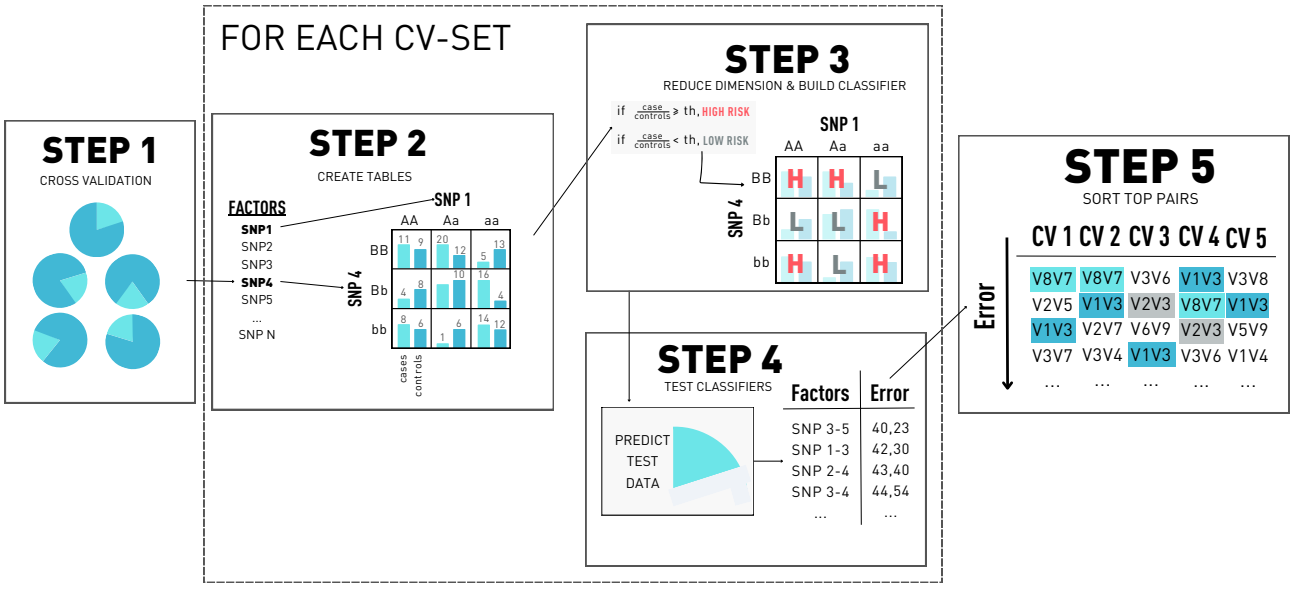


Figure 2: Multi Dimensionality Reduction (MDR) use case overview

3.4 Data Connectors

For these use-cases, we first needed to carry out an extended exploration of the computational problems to define best which were the real challenges we should target in the context of the project. Once this exploration was done we determined that:

1. Ingestion of data is the main challenge of the GWD use-case since it applies machine learning. The straightforward approach is to read all the data at once to train and produce a highly accurate model. However, due to the magnitude of our data (4TB), we can't handle it. Consequently, partition strategies must be applied. Applying machine learning to a partitioned dataset, however, is a complex task. The learning technique is comprised of a first training stage, using a big portion of the dataset, followed by validation with a section of the dataset not seen during the training stage. This validation stage compares how good the training is. As bigger the training dataset is, the better the accuracy. However one must be careful not to take a too big chunk of the dataset, so we can perform as well the validation stage with a significant portion of it. Moreover, we finally need a testing dataset not seen by any of the other stages, which needs to be big enough as well to obtain the final accuracy. This process is already complex on a single node, when do we need to partition the data and generate models on each of the partitions, this task gets more challenging as we need to decide which portions to apply on which node, which at the same time will need to decide the partition sizes for the training, validation, and test datasets. Thus, the more partitions we make, the more challenging deciding such appropriate sizes is. Finally, we need to merge all the models into one applying a final training, validation, and testing datasets. One must notice the dataset used all the time is a single one, so the initial dataset size and variety decide the amount of partitions one can make, and thus the ability to split the computation process across nodes while preserving accuracy.

2. Computational power: in the case of the MDR the main problem is not data rather than processing statistically all the pairs as fast as possible. Even with a small number of cases (500MB) the computational power may already require 1 hour to process due to the factorial growth of the pairs combinations.

3. Selection of the appropriate top pairs for MDR: this was a critical decision. This use-case is based on the statistics on which variants cause the diseases. Therefore a proper and deep exploration of the variants causing it was necessary before being able to apply such statistical reductions to the problem and we devoted most of the effort there.

Based on those conclusions we were able to tackle the challenge of how to optimize the use cases to expand its scalability in terms of data-throughput as well as computational time. Based on this we started the development of the following data connectors:

3.4.1 Multi Dimensionality Reduction (MDR)

As described earlier this use case is based on statistical methods to discover pairs of variants that contribute to the development of T2D. This use-case does not have a problem reading data but rather a computational problem, as per each variation pair the amount of processing required increases factorially. This use-case first loads the data in CSV format with the different patient features. This CSV as explained earlier comes from a synthetic dataset simulating the same structure as a real dataset. Thus the first connector developed was a CSV data loader for genomics data. Once this was done, we had the computational problem of computing all the possible pair variants as fast as possible. For this, we have developed an HPC data connector allowing us to optimize the usage of supercomputers. Initial tests show a speed-up of around 7x compared to a previous version implemented using Apache Spark. This connector directly communicates with the data predictor to determine if the selected pair combination is or not a potential case of T2D carrier. Summarizing the data connectors for MDR use-case are:

- HPC Data Connector: by using MPI libraries we can speed-up the computational process of the pipeline. We are seeking to request a large number of supercomputing resources and run real-sized experiments with larger synthetic datasets.

- Data Selector: selects which data (variants) will be used to analyze the pairwise combinations.

- Data Preparation: data connector that loads the data in CSV format with the patient features and its genome variants, and prepares it to perform the association between variants.
- Test of association: associates the different variants with clinic and control cases.
- Data functional interpretation: this connector gets as an input the amount of variants identified on the training sets and based on the known pairs causing the disease predicts whether or not the patient must be classified as a potential carrier of the disease. In this case T2D.

This connectors are summarized in figure 3. Figure 4 depicts moreover the two data association connectors developed for HPC (MPI) and Spark (traditional Big Data implementation).

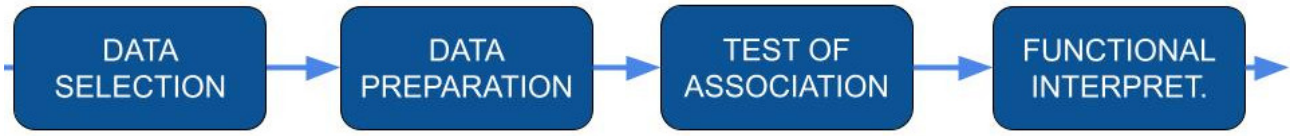


Figure 3: Data connectors used in MDR

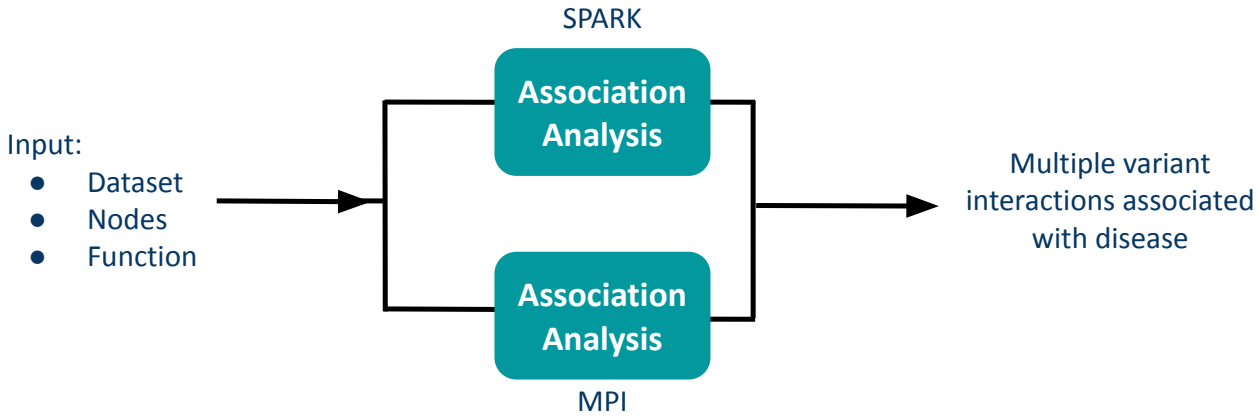


Figure 4: Association analysis with MPI and Spark connectors

This use-case is being optimized mainly using the HPC data connector which leverages MPI to speed up computation. Preliminary results show a speed-up of about 7x times concerning the initial implementation using Apache Spark. Further analysis of the gains is explained in the next section.

3.4.2 Genome-Wide discovery (GWD)

The Machine Learning Use Case consists of three steps: read, train, and test, which will be repeated in a combinatorial way over 112 million times. Notably, under the scope of the NearData project, these iterative tasks can be divided into the following connectors: Partitioner, Selector, Merger, Learner, and Predictor. Among the different approaches that can be applied to improve the performance of the models behind these connectors, during the first year of the NearData project we have been working with the Partitioner and the Learner. Despite the implementation of these connectors is still premature and under discussion, we expect its application to affect KP1 and KP3, thus involving significant performance improvements in data ingestion and scalability. More specifically, in the first tests performed with the implementation of the Partitioner, where the Lithops framework has replaced a sequential data ingestion process, we have increased the speed of the model in x1.88. Additionally, the update of the Learner connector to enable the use of GPUs by the Machine Learning

model has enhanced the use of all the available computational resources from MN4, thus increasing the speed by x1.06.

The connectors for this use-case are similar to the one of MDR but with significant differences among them. They are described as follows and depicted in figure 5:

- Data Partitioner: this component splits large data into small pieces that can be ingested at once. Through the integration of the Lithops framework, a sequential data ingestion process has been replaced by a flexible and scalable ingestion. This process splits the data into small partitions that allow to access each of the partitions quickly. To achieve this performance each of the partitions is given a identifier (using a hash function). With this partitioning, each of the nodes can quickly access and compute its partitions which will later be merged with the rest of the partitions. This partitioner has been developed to read data structured under VCF format.
- Data Selection: similar to MDR, select which data (variants) will be used to analyze the pairwise combinations. This process is done by each partition on each separate computational node.
- Data Merger: this connector merges the data retrieved from each of the partitions to compose a final model to feed the data predictor.
- Data Predictor: this component gets a model as input as well as data. Out of this data, the inference using the model. This component has been also sped up through the usage of GPUs, which allow a speed-up of x1.06 using MareNostrumIV accelerated partition. It is expected to improve this even further with more improvements allowing for multi-GPU utilization and new accelerated partitions.

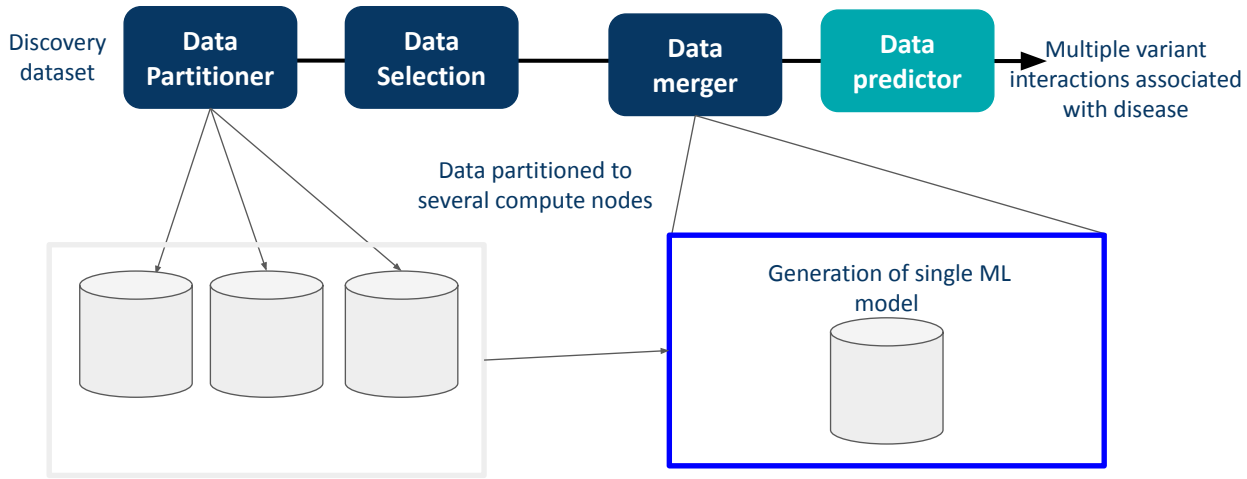


Figure 5: GWD use-case data connectors

This use-case is getting integrated with Lithops. Preliminary results show potential performance gains which are described in the next section.

3.5 What are the benchmarks/KPI?

In this section, we present the KPIs which evaluate the performance of both use cases. To do so we carry out a set of experiments utilizing the supercomputer infrastructure we have available at BSC, and when applicable we compare it as well with traditional cloud providers.

Currently, the benchmarks have been performed on a rather small amount of resources. This is because accessing a broader amount of resources available in the supercomputer special access requests must be made through the grant of a RES [2] or even a PRACE [3] proposal. The difference among them resides in the number of resources that can be granted. The PRACE grants the most, however, typically such large grants are made by splitting granted resources across supercomputers

of the network, instead of having them all in a single one. This, requiring to adapt the workload to run in different environments and then combine the results. An application for an RES has been already made and awaiting acceptance to carry on with large-scale experiments. Depending on the outcome of this initial large-scale experiments we may apply for a PRACE to expand the testbed even further.

3.5.1 Experiment 1: Multi Dimensionality Reduction (MDR)

The first step to being able to successfully run experiments on this use case was to produce a synthetic dataset out of the NuGENE cohort data. This was done by applying a Chi-Square test to all the pair combinations of the dataset, obtaining a significant value per pair. We use this value for filtering the non-significative pairs and reducing the dataset. This is depicted in figure 6

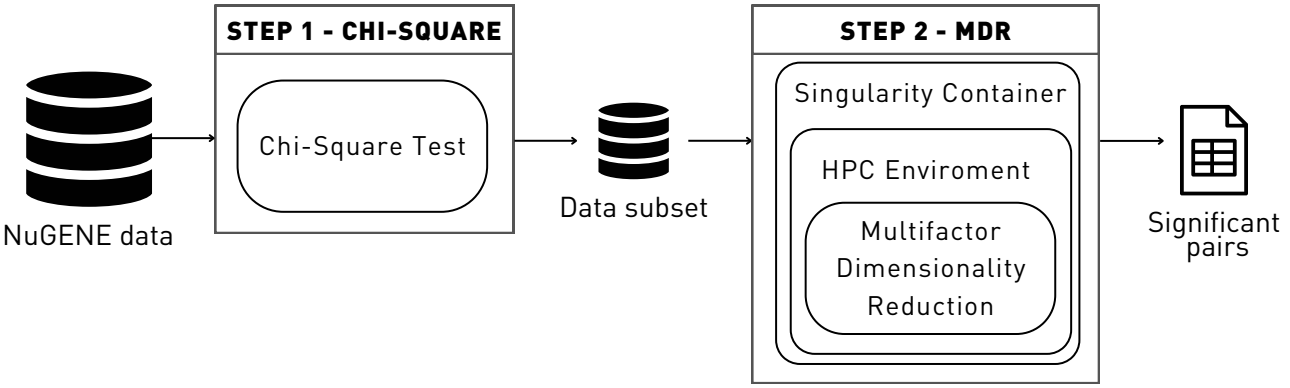


Figure 6: Chi-Square test dataset reduction for MDR

The MDR use case is implemented in singularity containers. This feature allowed us to run not only on our virtualized cluster but as well on supercomputing environments, allowing us to massively parallelize our workload. For this purpose, an initial version was developed using Apache Spark, HDFS, and Zookeeper, as depicted in figure 7.

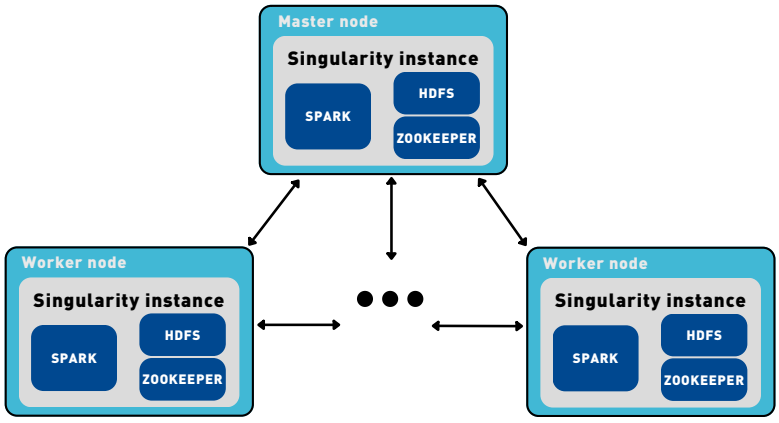


Figure 7: MDR containerized architecture

With this implementation, we have been able to process a million pair combinations in a distributed environment, for now experimenting with the scalability on up to 3 worker nodes (processing nodes). This infrastructure thus also contains a fourth node used by the Spark master to synchronize. Figure 8 depicts the computational time spent according to the number of workers nodes.

Following these results, which seemingly show linear scalability per number of nodes, we repeated the experiment utilizing larger datasets. We have increased the number of variants computed

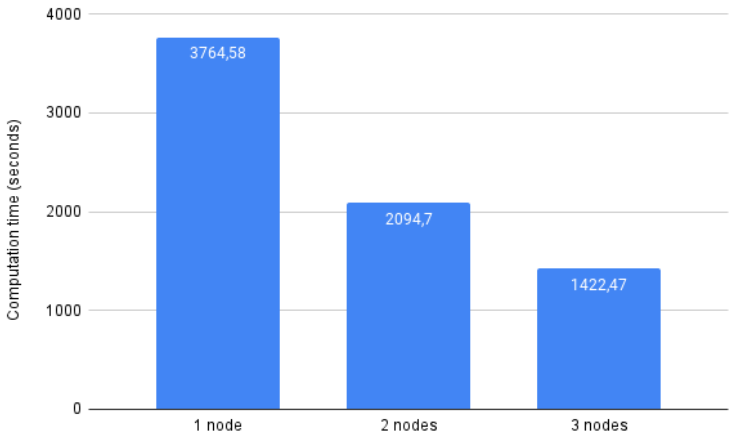


Figure 8: MDR scalability per nodes using pairs. Y-axes represent computation time and x-axes number of workers nodes used.

per million, from 1 million to 6 million. We have observed there is a perfect linear scalability per node. Each pair is processed independently and thereafter these results were expected. However, despite this scalability processing the total 15 million variants and the $6 * 10^{13}$ combinations would take almost 100 days to complete in a supercomputer such as MareNostrum4 (MN4).

This is just one step of the process, though. The next step is to decide the top predictor variants that will be used to apply our method over the 5 cross-validation sets we are using. Moreover, ideally, we would like to use 10 cross-validation sets, but once again we are dealing with an extreme-data problem, and using 10 sets would be too computationally expensive. To reduce the input dataset we have applied the Chi-Square test as described above. Although this test is computationally less expensive than the MDR, we still have to deal with a huge volume of pairwise combinations. Using MN4 and the Greasy software[4] to distribute more than 2000 parallel tasks across 16 working nodes (48 cores and 96 GB of RAM) we have tested the association of 11.297.253 variants in less than 4 days. For MDR we have kept the pairs with a relevant association with the diseases. The selected associations are those with a p-value $< 1 * 10^{-6}$, corresponding to 1.883.192 pairs. Figure 9 shows the distribution of the Chi-Square pairwise interaction p-values depending on the MDR consistency value, which corresponds to the number of times a pair appears at the top pairs of the CV dataset. Consistently, the total number of pairs decreases after each step of the CV, ending with only a few pairs that are detected to be relevant and independent of the dataset partition.

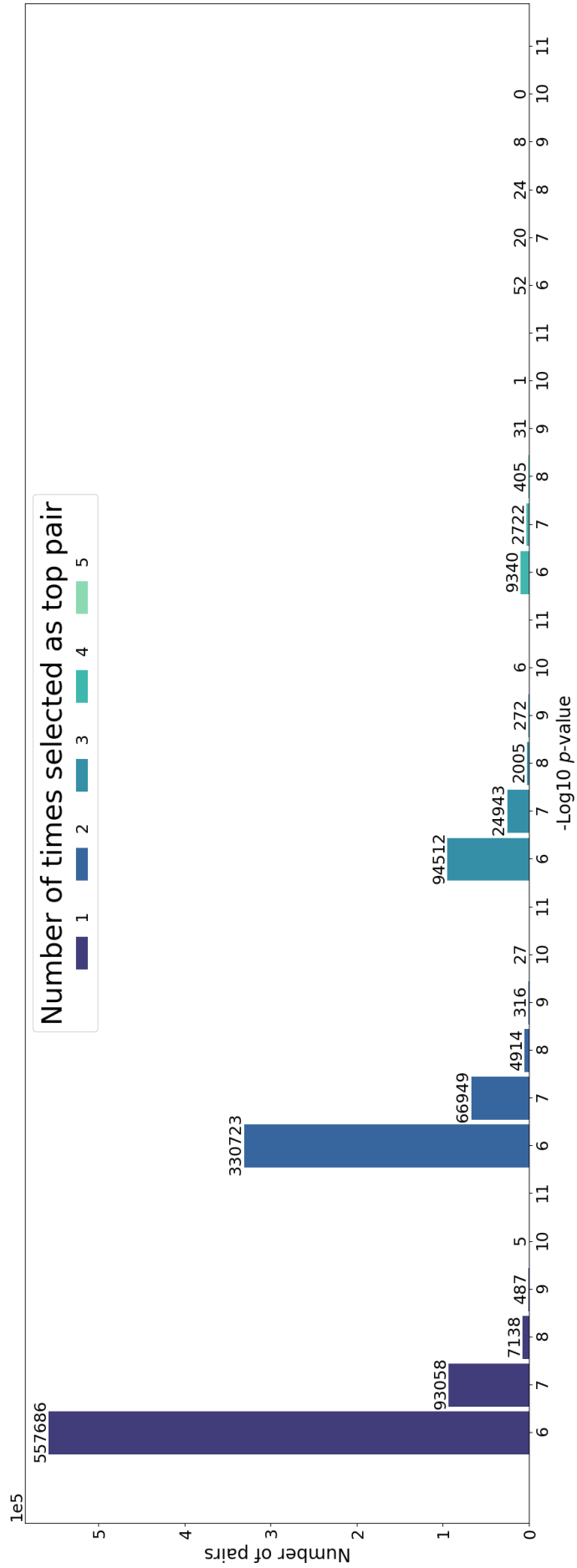


Figure 9: P-value distribution in logarithmic scale for the studied pairs, grouped by the times they have been selected as top pairs from the 5 CV dataset.

Finally, we have made the final selection by setting a hard filter for keeping only the pairs that appear in all 5 CV datasets as the top 20% predictor pairs. From the 1883192 pairs analyzed, we have obtained a set of 104 pairs. These results suggest that the discovery of the interacting pairwise will depend on the calculated effect size that the variants will have with the disease, independently of how much lower the p-value is. Regarding the significance level of the top 104 pairs, we have measured the prediction power of the top pairs from the MDR using the average precision (see Figure 10) of the 5 CV sets. As expected, the group with the best average precision (60,17%) corresponds to the group of top pairs common across all the CV sets. However, some pairs have a higher average precision in other groups. This can be explained by over-fitting in that set, meaning that its relation with the disease is completely related to the data analyzed.

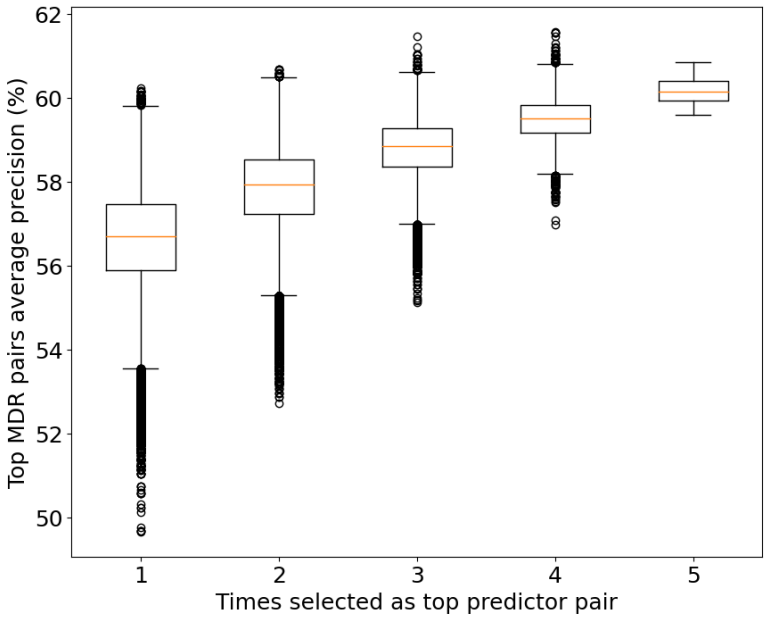


Figure 10: Box plot of the average prediction power (precision) for the studied pairs and their consistency value. To be selected as a top predictor in each CV set, a pair must be inside the 20% top predictor pairs of the set.

According to the state of the art on T2D, we found that the selected top pairs are known to be associated with the disease and hence proved the biological value of our method.

MareNostrum 4 supercomputer infrastructure

As explained earlier, this use case is mostly focused on computational power. Consequently, this use case is the most suitable to enhance it leveraging an HPC data connector. In NEARDATA we have used MareNostrum 4 supercomputer to demonstrate such a connector. Despite this supercomputer brings great computer power, it has its limitations when it comes to usage. A significant limitation is the fact that being a production environment, and a shared one implies security constraints such as not using docker containers because they inherit from the init process. Consequently, they are subject to potential security leaks. Similar software is also faced with such limitations. Importantly, all software pretended to be installed must be done so either at a user-level or ask for the operators to do so. In the latter case, the software will be carefully examined and any feature not allowed will be stripped and not installed. This process, on the other hand, is a long and tedious process that may require weeks to install.

When it comes to the hardware itself, MN4 contains 3456 nodes with 48 cores each (2 CPUs of 24 cores). Each of the CPUs is a Xeon Platinum 8160 operating at 2.1GHz. The main memory is of 96 GB per node, with 32 KB and 1 MB of private L1 and L2 caches, respectively. And a shared 33 MB L3

cache. The software environment works on top of a SuSE Linux Enterprise 12.2 operating system.

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).
We have observed that our method scales linearly by the number of computational resources available. We can safely say it is a computational problem rather than a data ingestion problem. That is, we do not necessarily need to have a fast ingest of the data since just reading a minimal amount of data requires a processing time over passing the time required to read the next block from disk. With this idea in mind, our method was re-implemented using MPI to maximize the computational power a supercomputer such as MareNostrum can provide.

We made a comparison running Spark on MareNostrum as well as our implemented MPI method using the HPC Data Connector. We observed how the new HPC version overpasses the capabilities of Spark and observed a significant speed-up of 5x. This is depicted in figure 11.

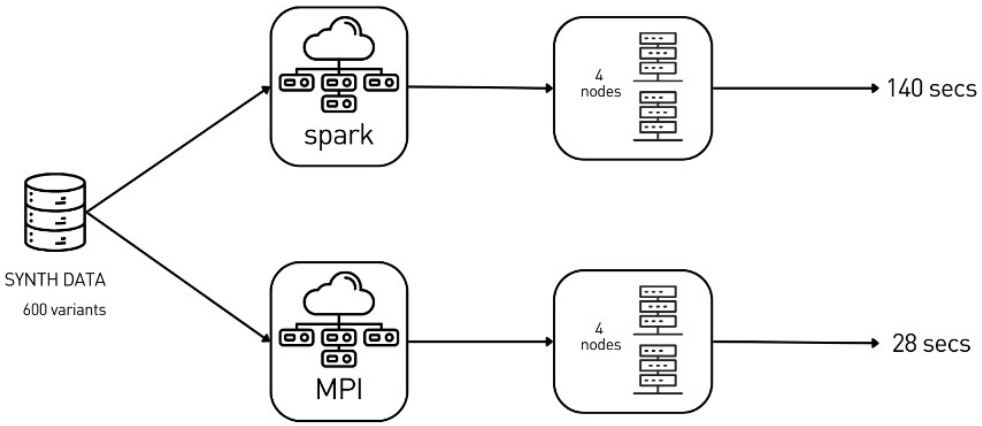


Figure 11: Execution time comparison of MDR implementations with Apache Spark versus HPC in MareNostrum 4 utilizing 48 cores.

Following this initial evaluation, we run this version of the use-case with different amounts of cores per node, nodes, and combinations to compute. The goal was to determine the best ratio of combinations to compute per core and second. The challenge towards this goal was that MareNostrum has a vast amount of cores and nodes available, making it non-trivial to find the best combination of pairs to process per core and second. Thus, we started studying the best configuration possible and the results are shown in figure 12. The size of the dataset determines the amount of pairwise combinations to be processed. Out of it, we split the dataset into several batch sizes per each core to process. Each of the nodes has 48 cores available. The x axes show this batch size to process per core, while the ya axes show the combinations processed per second and core. The bars legend indicates the number of combinations to process per file (s50 stands for 50x50 combinations per file while s200 is 200x200 combinations per file). The number indicated the aggregated amount of cores used (which is a multiple of 48 as it is the amount of cores available per node). We have tested on up to 8 different nodes.

Conclusions: for this experiment our efforts have gone in one hand into determining which are the most relevant pairs to analyze. This selection is especially relevant as it defines how well our heuristic can perform. On the other hand, we have devoted many efforts into the creation of a new HPC data connectors that allows moving complex computational processing into HPC framework such as MPI. While this connector shows potential we faced the new problem of deciding which was the best amount of combionation of variants to process per core and per node. This decision was

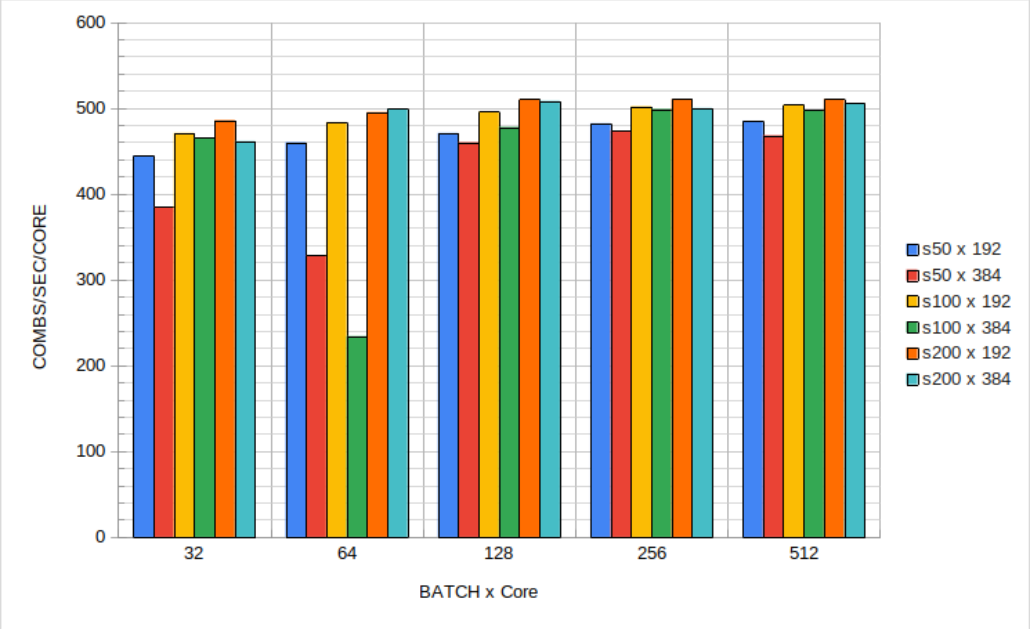


Figure 12: Chart showing the best set-up of combinations to process per core and per node.

essential in order to effiicently compute in a large number of nodes. Now that all this work is carried out, through the grant of a RES [2] we expect to be able to carry out large-scale experiments by M18 of the project.

3.5.2 Experiment 2: Genome-Wide Discovery (GWD)

This use-case poses the challenge of ingesting a large amount of data to train and validate a machine-learning algorithm. Currently, as earlier explained, we are targeting only pair-wise combinations. However, in an ideal world, we would like to compute all the combinations in sets of 23 (considering all the chromosomes). Ingesting all the data in a single node is not possible, not even in a supercomputer. The main challenge we are facing is to ingest, partition, and distribute data as fast as possible. Consequently, we employ a distributed learning method to cope with vast memory requirements and to reduce training time.

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).
Regarding this use-case, we are exploring the different research challenges to improve the data throughput (ingestion of data) of the pipeline:

- **ML cross-validation**: The use and application of Machine Learning (ML) models require the employment of diverse protocols to ensure the robustness of the predictors and their reliability. To confirm the applicability of the methodology developed under the GWD use case to solve the variants interaction problem, we have expanded the analysis and replicated the results in a completely independent cohort. To that extent, we have conducted 2 steps: 1) training and test in the 70KforT2D dataset followed by a prediction in the UK Biobank, and 2) training and test in the UK Biobank followed by a prediction in the 70KforT2D.
- **Lithops ingestion**: ML models are usually encoded with 3 steps in a sequential manner. These steps involve 1) a data ingestion process, 2) training, and 3) testing. Particularly, this sequence

is expected to be applied once, which means that one dataset will be only ingested, trained, and analyzed only one time. However, in this particular case, the diverse computational limitations that encompass the analysis of variants interaction, justify a combinatorial analysis of the data, which involves multiple analyses of thousands of fractions of the initial dataset and, therefore, multiple ingestions, trains, and tests from the same dataset. To enhance the ingestion of the data and ensure the best performance, we adapted this process to use Lithops, a framework for big data analytics that facilitates the fraction and ingestion of the input dataset in a parallel manner. The use of this tool improves the performance in 2 directions: 1) It ensures a unique ingestion of the initial file, thus avoiding the time consumption of multiple reads of the data, and 2) Its parallel architecture enhances the ingestion time.

- **GPU + CPU**: Supercomputer architectures focus on performing fast operations and, therefore, are usually based on CPUs and GPUs. The preparation of the multiple functions of the ML model to be applied in both CPUs and GPUs ensures the ability of the methodology to be applied in any architecture. The creation of connectors that facilitate the use of CPU and GPU promotes the better use of the available resources, and the auto-scaling to diverse computer architectures. Current tests have shown an speed-up of 3.1x times respect CPU-only, tested on the accelerated partition of MareNostrum IV utilizing a single NVIDIA Volta V100 GPU. Thus showing great potential considering there are up to 4 GPUs available in such nodes.

KPI-3: Demonstrated resource auto-scaling for batch and stream data processing, validated thanks to data-driven orchestration of massive workflows.
Moreover, this use case also targets to improve KPI-3 via enabling autoscaling of the application with the integration of Lithops data connector. The advances in genomics during the last decades have promoted the possibility of the analysis of more samples and, therefore, the creation of bigger cohorts with hundreds of thousands of individuals, and even millions. This new era has enhanced the discovery and the progress towards a better comprehension of the genetics behind complex disorders, but the big computational demands to analyze this huge amount of information still represent a challenge that involves, among others, the use of auto-scaling methodologies. Here, to provide a robust data ingestion connector that ensures the auto-scalability of the problem, we have implemented Lithops. Particularly, Lithops ensures the parallelization process of the data ingestion process, securing the ability of the connector to be applied in large datasets with more individuals and more features.

To be able to perform real experiments we needed to have the integration with Lithops completed in order to run the use case on MareNostrum. This integration is described in the next section as well as some baseline evaluation of the integration performance. Taking advantage of the same RES already requested, large-scale experiments will start. We do target to complete them in the next reporting period.

Conclusions: we have devoted most of our efforts in being able to ingest much more data that we can now. This is an essential aspect of this experiment, as the larger the data utilized the better the trained machine learning model will be. In order to do so we targeted the integration with the Lithops framework, as one of its main goals it is precisely to efficiently distribute data across a large number of nodes. This integration task is described in detailed in the following section. On the other hand we have also made an effort on enabling GPU usage for our implementation so we can expand our HPC data connector to utilize GPUs and move complex vector computations to such specialized devices. In the following period we expect to be able to complete this work and carry out large-scale experiments through the grant of a RES [2].

3.6 Integration with the architecture components of NEARDATA

Lithops framework is being integrated into the GWD pipeline to improve scalability. GWD pipeline has a data ingestion problem, as it requires reading a vast amount of data which will be used to apply a machine learning model. Due to the quantity of data, a distributed learning model is utilized. Thanks to Lithops we can scale this distribution phase and be able to ingest more data than

before. One of the main challenges was that due to confidentiality reasons the data is only available on MareNostrum supercomputing, hosted by BSC. As explained earlier, the supercomputer is a production environment with security restrictions, such as not being able to run docker containers. The Lithops framework backends were currently making use of docker containers, either directly or through platforms such as Kubernetes. Neither of those platforms can work in this production environment.

Consequently, during the integration phase, we have initiated the development of a new Litohps backend utilizing RabbitMQ and POSIX filesystems as storage. The fact that we are now supporting a POSIX filesystem means we can also take advantage of the MareNostrum storage system, GPFS, which allows for a greater storage bandwidth compared to traditional storage. This backend allows to run Lithops on the supercomputer, as our RabbitMQ setup does not require privileges to run. On the other hand, reading from the filesystem allows us to make use of the GPFS storage in place on the supercomputer.

Figure 13 shows the above changes for Lithops to use singularity containers to address the security issues presented by the MN. Each machine in our cluster contains a Singularity container, which allows the execution and management of all the necessary Lithops code to launch the N necessary functions, taking into account these security restrictions. These functions are distributed equitably among all the machines in the cluster by receiving different packets through RabbitMQ.

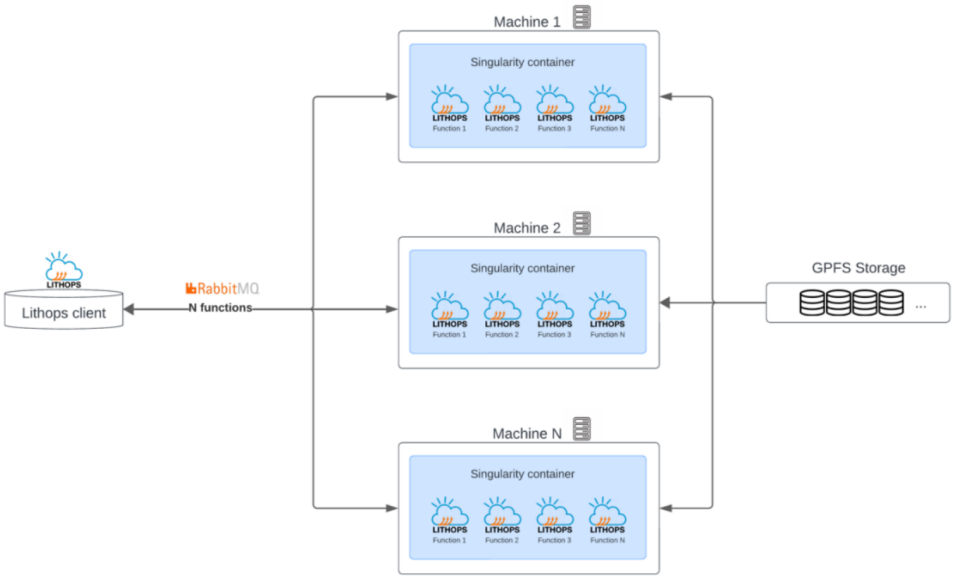


Figure 13: Architecture of novel Lithops compute backend using singularity containers.

These packets contain the functions to be executed, so each container, upon receiving a packet, checks if there are available resources and if there are, it launches as many functions as possible. In this way, the nodes themselves manage the resources through the Rabbit work queue, ensuring that all available resources are always used, avoiding wasting CPUs.

This work queue system allows for much faster and smoother communication between the client and the workers. Furthermore, it allows for customization of these packets so that the available resources can be managed as efficiently as possible.

Once this new compute backend was designed, the same was done for a new storage backend. In this case, a storage backend was designed to work on the system's file system. This was necessary since until then, there was no way to work with GPFS. This new backend allowed for read and write speeds as high as the disk used would allow.

This integration grants elasticity and flexibility in HPC environments which are typically not. And at the same time we get the full power of computational nodes designed to maximize compute

power. Lithops is designed to run in the cloud and especifically in FaaS environments, providing great elasticity and flexibility at the expense of less computational power. Thus, we compare what we achieve by running it on the supercomputer versus in typical FaaS environments using Lithops FLOSP benchmarks. The results show how MareNostrum improved the performance with respect to the current FaaS benchmarks tested in both execution time and average FLOPS as depicted in figures 14 and 15 respectively.

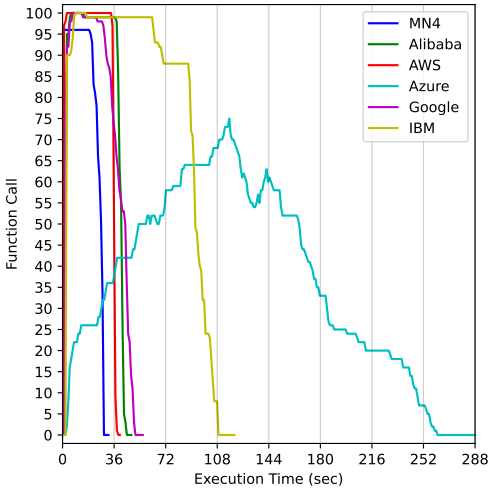


Figure 14: Performance in terms of execution time running the FLOPS lithops benchmark on top of MN4 compared to commodity FaaS providers.

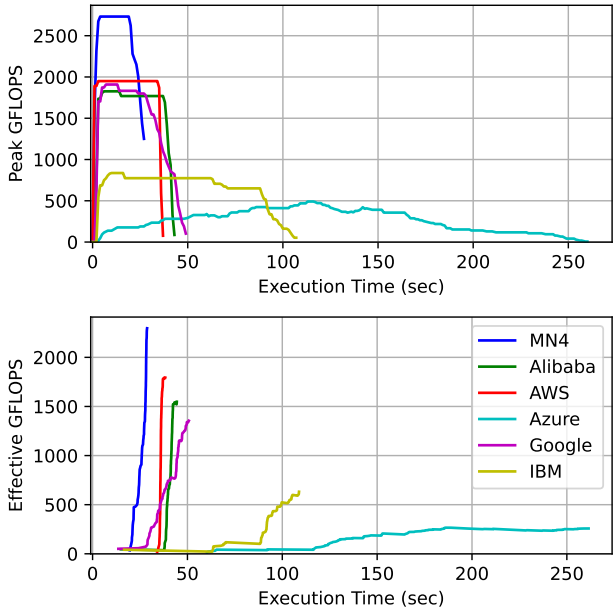


Figure 15: Performance in terms of GFLOPS running the FLOPS lithops benchmark on top of MN4 compared to commodity FaaS providers.

Figure 16 shows the performance of the storage backend. This picture compares the aggregate read and write bandwidth per function on the different cloud providers versus MN4 (blue). This presents rather poor write results because the use of GPFS in MareNostrum is shared among all machines and several users are running concurrently. It is not possible in such an environment to re-

quest sole use of the entire supercomputer as it is a public infrastructure intended to advance science in many aspects. So it is of utmost importance to share it as much as possible while considering the performance trade-off. Future next steps include examining how can we deadl with this situation to improve storage performance.

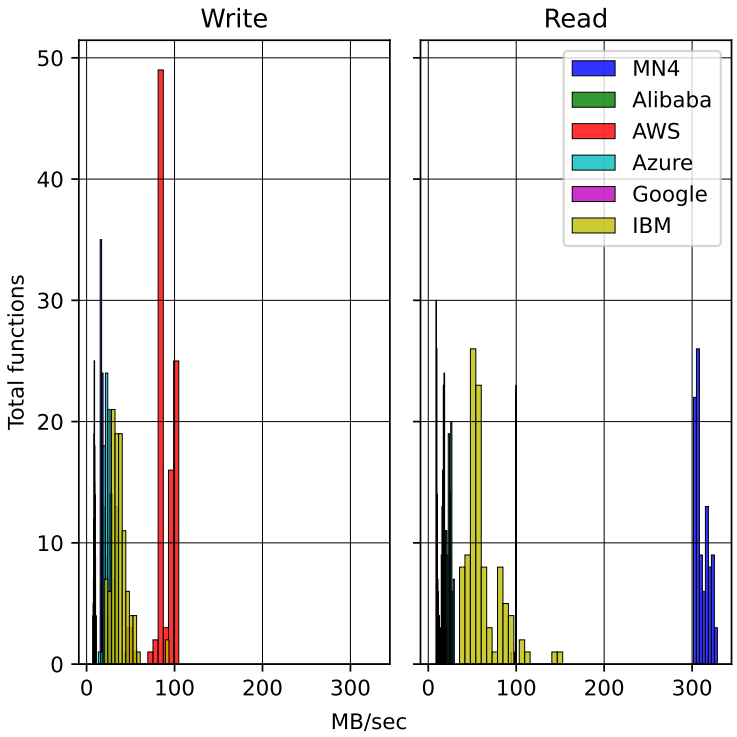


Figure 16: Lithops Storage benchmark with MareNostrum.

After this integration we are now able to launch and execute Lithops efficiently and quickly on HPC systems, in this case, on Marenostrum. This allows resource management, such as CPU and memory, to be fully managed by this tool, enabling developers to save efforts in such tasks. This implementation represents a significant innovation for Lithops and these massive computing systems.

4 Use Case Surgery

4.1 Why is the use case extreme data?

Recent advancements in surgical technology have yielded vast amounts of valuable data, holding immense promise for improving patient care. Yet, the diverse range of data sources presents a significant challenge for surgeons, particularly within the operating room. In surgical scenarios, the concept of data-driven computer-guided surgery (CGS) emerges as a solution to aid surgeons during procedures. Specifically, in laparoscopic and robotic surgeries where challenges like depth perception and hand-eye coordination arise, CGS provides assistance functions such as tumor localization and predicting surgical complications by analyzing the surgical workflow through endoscopic camera video streams.

However, CGS systems encounter various challenges due to the large volume of data they handle. A robust CGS framework must effectively manage:

- Large volumes of real-time video data.
- High-resolution data streams.
- Temporal information (real-time analytics) crucial for meaningful guidance during surgery.

Moreover, owing to the inherently sensitive nature of patient data, stringent regulatory compliance and confidentiality requirements come into play. Thus, the utilization of confidential and secure computing technologies becomes imperative to ensure a safe, protected, and trustworthy execution environment.

4.2 Assessed Datasets

Throughout this project, our primary focus will be on utilizing in-house data 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) [5].
- HeiChole (University Hospital Heidelberg and NCT/DKFZ) [6].
- Cholec80 (University of Strasbourg) [7].
- Synthetic video data (NCT/DKFZ) [7].

Moreover, given our close ties to the surgical facilities located on the university hospital campus, we aim to expand our dataset collection to encompass a broader spectrum of surgical workflows.

4.3 Description of the use-case

Our use-case in surgery encompasses two distinct focal points. Firstly, we address the challenges inherent in Federated Learning, with a primary objective of improving its security protections. To achieve this, we leverage the Flower framework, enclosing its functionalities within a Docker container for streamlined deployment and management. Additionally, we integrate Scone to ensure security, ensuring the preservation of privacy throughout the Federated Learning processes.

Secondly, we embark on the development of a surgical video streaming application to handle multiple inference jobs. Our current testbed on Dell's Cork lab can be found in Figure 17. Our strategy revolves around the integration of Pravega and GStreamer into our existing pipelines. This integration empowers us to harness the advanced processing capabilities offered by GStreamer plugins, including our developed segmentation, phase detection, and tool detection plugins. Furthermore, by integrating both GStreamer and Pravega into our infrastructure, we establish a robust ecosystem that contributes to efficient data collection, processing, and storage within the context of surgical video streaming.

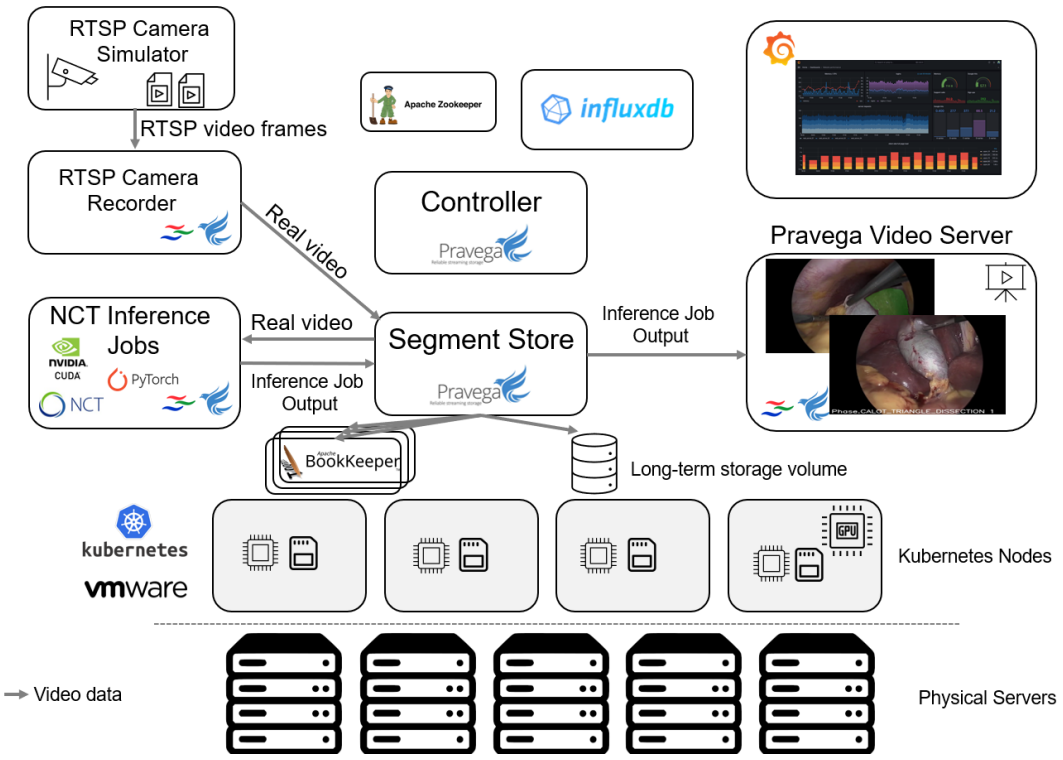


Figure 17: Deployment of Pravega and NCT video stream analytics in Dell's Cork lab.

4.4 Data Connectors

4.4.1 Federated Learning

We constructed a Docker container to leverage Flower as a Federated Learning framework for our experiment. This setup streamlines the establishment of distributed training for the models we aim to train. Docker enables us to efficiently integrate Scone's capabilities to enhance security measures, ensuring the protection of personal information within Federated Learning processes.

4.4.2 GStreamer and Pravega

Our developed data connectors for Pravega and GStreamer are implemented as GStreamer Plugins, integrating seamlessly into the system. Pravega, serving as the streaming storage solution, efficiently manages the storage and retrieval of video streams. Concurrently, GStreamer processes and analyzes incoming streams in real-time using data connectors. Our Plugins incorporate machine learning models. The plugin is responsible for extracting current images from the Pravega video stream and forwarding them to the model for analysis. Furthermore, the results of this analysis are then overlaid onto the incoming video stream and published into another stream. Depending on the task at hand, different models are utilized to ensure optimal performance.

The following methods have been developed and integrated into plugins:

- Semantic Segmentation of the liver:
  - Assigns a class value to each pixel of an image, indicating whether the pixel is part of the liver or not.
- Phase Detection of Cholecystectomy:
  - Classifies the current frame into the corresponding step of the gallbladder removal procedure.
- Surgical Tool Detection:

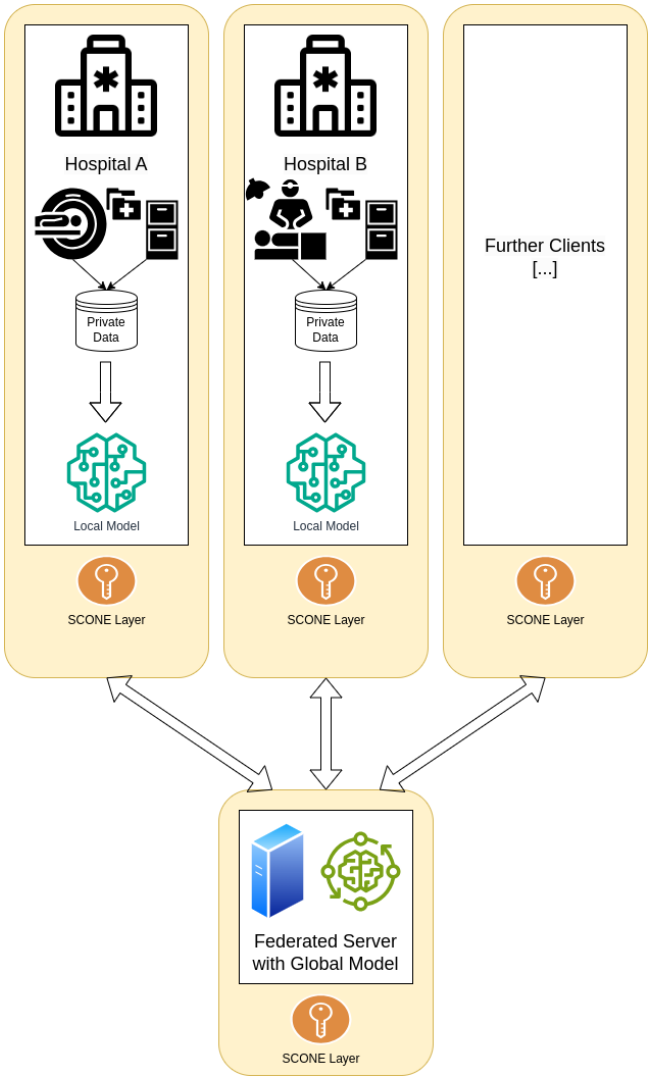


Figure 18: Secure Federated Learning Pipeline leveraging SCONE

– Determines the presence of a surgical tool in the current frame and identifies its type.

4.5 What are the benchmarks/KPI?

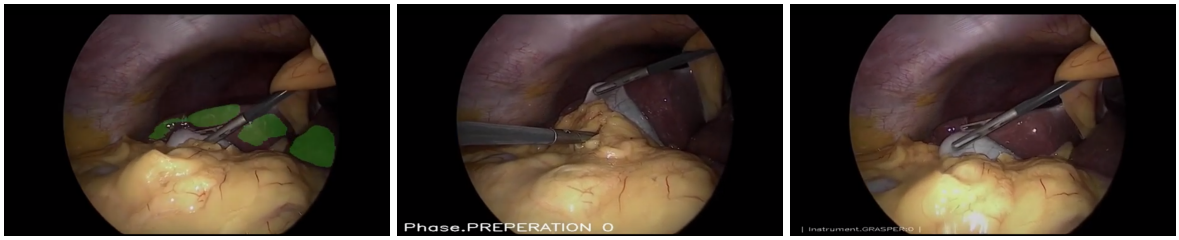
4.5.1 Federated Learning

KPI-3: Demonstrated resource auto-scaling for batch and stream data processing, validated thanks to data-driven orchestration of massive workflows.

Machine learning algorithms are often trained in a centralized manner, where all necessary data is collected from various sources and stored on a single server for local training. However, in the medical field, strict data regulations prohibit direct data transfer without approval from the ethical committee.

Decentralized training strategies, such as Federated Learning, offer an alternative approach. In Federated Learning, data remains distributed among partners, and each partner trains their model locally. These trained models are then shared with a central server, which aggregates them into a single global model. At the end of each iteration, the global model is sent back to the partners. This approach allows for a well generalized high performance across all available datasets.

Federated Learning demonstrates the ability to process more data compared to centralized learning when data sharing is permitted due to the fact that the model instead of the data is shared. More-



(a) Semantic Liver Segmentation (b) Phase Detection of Cholecystectomy (c) Surgical Tool Detection

Figure 19: **Surgical Image Analysis:** This figure showcases three images representing different aspects of surgical image analysis. The first image demonstrates semantic liver segmentation with a green overlay. The second image focuses on phase detection in cholecystectomy, and the third image highlights tool detection. Both visualizing the output as text overlay. All of these techniques offer valuable insights and aid in medical decision-making.

over, the Federated Learning framework, Flower, streamlines the addition of more partners into the training process without requiring additional effort. This enables auto-scaling of the training process. Additionally, Federated Learning can lead to improved model performance compared to individual models trained on separate datasets.

To sum up, we want to combine the federated learning platform, Flower, with Scone to train multiple clients with extreme data to achieve a high performance with a good generalization ability.

KPI-4: High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.

In addition to facilitating collaborative training, Federated Learning also ensures the confidentiality and security of the training procedure. This is achieved through the incorporation of our added SCONE environment.

SCONE will protect the execution, via the TEE in the server; i.e. the CPU and memory reading/writing. The assertion of the SCONE's efficacy is done by **inspecting the memory**. Linux handles memory as a representation in the filesystem in the form of files. For example, assuming the process running federated learning has a process id (PID) **1310177**. We will, in turn, read two files that the operating system uses to handle memory: the `mem` and `maps`. So, for this running process, the files that will be read are `/proc/1310177/mem` and `/proc/1310177/maps`; while at the same time their contents will be dumped to a third file created and owned by the adversary making the inspection.

The second phase of the assertion requires that we, systems administrators – in the role of the adversaries –, know some secret that resides persistent for as long as the process **1310177** is up or any particular piece of data processed during the time of dumping. With a simple `grep` or `grep -o` on the dump we can verify if that piece of confidential data is exposed. For example, we know that "**SURGERY**" ought to be in the dump of a non-confidential execution, hence `grep -o SURGERY memdump-confidential.log` should not bring anything.

Experiment 1: Execute the federated learning demo inside the SGX enclaves using SCONE's network shield, configured through the Data Broker's CAS(Configuration and Attestation Service) component.

Goal:

- Investigate how confidential federated learning application, leveraging Trusted Execution Environments (TEEs), address inherent challenges within the Federated Learning paradigm. These challenges include scenarios where attackers with privileged/root access could potentially access training models, instances where numerous malicious clients might collude to reveal local data and models of other clients, and situations where malicious clients could manipulate their

local training data or parameter updates forwarded to the central training system, thereby compromising the integrity of the global model.

- Evaluate the performance of the confidential federated learning application by examining how SCONE's network shield, configured through the Data Broker's CAS component, impacts execution time compared to both the Vanilla (Intel/Non-SGX) environment and the SCONE (Intel SGX/hardware mode) setup, with a specific emphasis on FL execution time.
- Evaluate the effects of optimizations on the startup latency of functions within the FL application architecture.

Observations:
Experimental Setup Experiment Setup We executed the performance evaluation experiments on a system with an Intel Xeon® CPU Silver 4314, 2.40 GHz processor, 128 GiB of main memory, and a 500 GB SATA-based SSD. All experiments are conducted inside Docker containers (version 20.10.19) using Ubuntu 20.04.
Scenarios

Scenario	Description
Vanilla (Intelno SGX)	The FL application executing on Intel CPUs without SGX support.
Scone (Intel SGX/hardware mode)	The FL application executing on Intel CPUs with SGX support, benefiting from all hardware-backed security guarantees provided by Intel SGX and SCONE.
Scone (Intel SGX/hardware mode) with Data Broker's CAS component	The FL application executing on SCONE integrated with CAS (Configuration and Attestation Service), several crucial security features are employed to ensure the integrity and confidentiality of the process. Network Shielding: SCONE's Network Shielding feature is instrumental in protecting network connections between the client and server components of the federated learning application. File Shielding: In federated learning, training data privacy is paramount SCONE's File Shielding capability encrypts the input training data using the file system shield, ensuring that it remains protected from unauthorized access.

Table 1: Scenarios

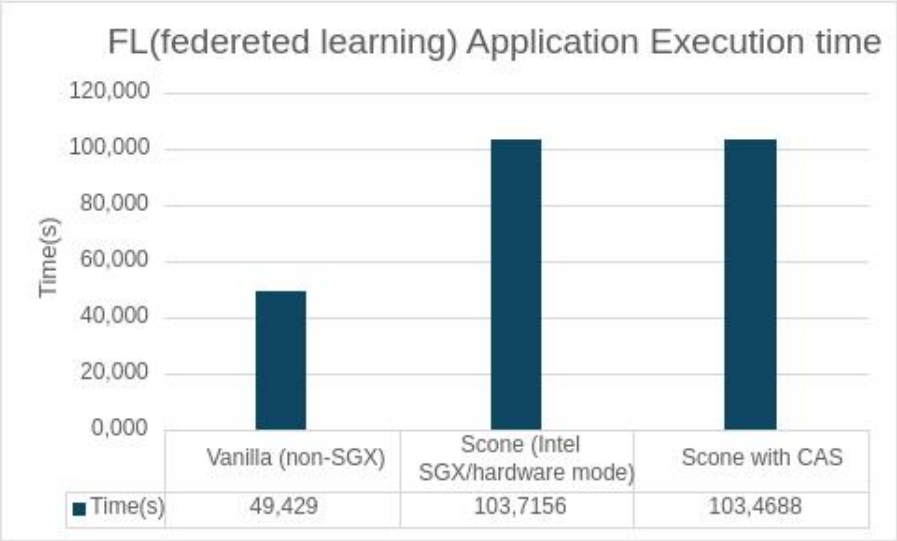


Figure 20: FL Execution time

The results Figure 20 illustrated that the overhead in execution time observed for Scone scenarios compared to Vanilla(nonsgx) corresponds to the enclave startup process. This process takes a fixed amount of time given an enclave size. Therefore, for applications that are more complex and take longer to complete, this overhead should correspond to a small fraction of the overall execution time.

Furthermore, SCONE uses two shields to ensure the security of data and network connections. The network shield is configured through the Data Broker's CAS component and protects network connections between clients and servers while controlling data access. The file system shield mechanism encrypts the input training data using SCONE's file system shield and decrypts it for processing inside SGX enclaves.

Function Startup Optimization

Function startup optimizations to evaluate the impact of our SGX-based optimizations on the function startup time. To explain these results, we conducted an additional experiment (Table 2), where we measured application startup time while varying heap sizes. SCONE_HEAP: size of the heap allocated for the program during the startup of the enclave.

Heap Size	Startup time
6GB	220.6911s
8GB	210.1497s
10GB	209.1326s
12GB	193.6606s

Table 2: Startup time of a no-op enclave depending on the dedicated heap size for SGXv1

In this table 2 the results showed that enclave startup time depends linearly on the heap size in the SGXv1 case. With larger heap sizes initialization time can reach 193 seconds (for 12GB heap), and it dominates the total application runtime.

Conclusion: The solution encrypts input training data and code, such as Python code, and performs all training computations, including local training and global training, inside TEE enclaves. This secure federated learning allows for all gradient updates via Transport Layer Security (TLS) connections between the enclave of clients and the enclaves of the central training computation. As a result, attackers with privileged access cannot violate the integrity and confidentiality of the input training data, code, and models. The Data Broker component that is integrated with CAS (Configuration and Attestation Service), based on the remote attestation mechanism supported by TEEs. This

component ensures the integrity of input data and training code. In other words, it ensures that training computations are running with the correct code, correct input data and have not been modified by anyone, such as an attacker or malicious client. The preliminary evaluation shows that we can ensure the confidentiality and integrity of federated learning computations while maintaining the same utility/accuracy of the training computations.

KPI-5: Demonstrated simplicity and productivity of the software platform, validated with real user communities in International Health Data Spaces.

In ensuring the productivity and user-friendliness of our project, several key questions must be addressed:

- Handling Multiple Large Datasets in Training:
  - Strategies for efficiently managing and processing multiple large datasets during training need to be explored. This may involve implementing data batching techniques, optimizing data pipelines, and leveraging distributed computing frameworks.
- Impact of Sconification on Execution Time:
  - Assessing whether the Sconification process significantly increases execution time is crucial. To mitigate any potential slowdowns, optimizations such as streamlining the Sconification process, optimizing code for better performance within the Scone environment, and leveraging parallel computing techniques can be considered.
- Incorporating GPU into the Scone Process:
  - Integrating GPU acceleration into the Scone environment can enhance computational performance which is important for machine learning. This may involve exploring techniques such as GPU passthrough or containerization to enable GPU utilization within the Scone environment. Additionally, optimizing algorithms and frameworks to leverage GPU resources effectively is essential for accelerating the training process.

4.5.2 Pravega and GStreamer

Next, we focus on analyzing the IO performance of Pravega for the NCT computer-assisted surgery edge video analytics use-case. In particular, our goals are:

- To analyze the IO performance of Pravega depending on multiple aspects such as writer/reader parallelism, video bitrates, network location, and underlying hardware performance (KPI-2).
- Evaluate auto-scaling mechanisms we built in Pravega for our computer-assisted surgery use-case (KPI-3).
- Estimate the simplicity and productivity improvements for NCT from using containers with Pravega GStreamer plugins installed and Pravega as a mechanism to automatically move video data to a long-term storage tier (KPI-5).

KPI-2: Significant data speed improvements (throughput, latency) in storage & access, and real-time analysis of, videos and sensor data.

First, we focus on evaluating performance improvements from using Pravega in the NCT computer-assisted surgery use-case. To this end, we have developed a performance evaluation framework for measuring IO performance of video streams in Pravega based on key metrics that impact user experience (*i.e.*, end-to-end latency, historical throughput). We performed a battery of video analytics experiments with Pravega in Dell's Cluster in Cork Lab (see Fig. 17). The cluster used for testing

was using Dell PowerEdge R750 servers with Intel Xeon CPU's using VMWare Virtual Machines. Each VM was given 8 vCPU cores, 16GB of RAM, and 250GB of local server NVMe SSD storage. A 100TB NAS server was also used to provide NFS remote storage for performance evaluation. A set of 4 VM's were connected using Kubernetes, with Pravega deployed within the cluster. To exercise our benchmarks on an alternative infrastructure, we have deployed our experiments on AWS Elastic Kubernetes Service (EKS). The cluster is formed by 3 EC2 `i3en.2xlarge` instances (acting as Kubernetes nodes) with 2 local NVMe drives each. We used the Rancher Local Volume provisioner [8] plus some scripts to provision local volumes for Bookies, which are the component in charge of the write-ahead log for Pravega. In both cases, we deploy Pravega as follows: 1 Zookeeper instance, 1 Pravega Controller instance, 1 Pravega Segment Store instance, and 3 Bookkeeper instances. In the Cork on-premises cluster Bookkeeper writes data to both network storage volumes and local drives. Pravega is configured to use 1 data replica in Bookkeeper per event stored, which we is good enough for video analytics use cases. We use for comparison the latency of the AI inference jobs in Fig. 19.

We focus on performance metrics that have a direct impact on the customer experience when deploying video analytics applications:

- *End-to-end latency*: This metric, measured in milliseconds, accounts for the time taken for a writer to durably write a video frame in Pravega plus the time taken for a reader to read that video frame back. Note that each video frame (or event) stored in the system is durably stored in persistent media before acknowledging the writer, as data durability is one critical guarantee that the system provides. The time calculation is done by time-stamping video frames at the writer side and calculating the delta with the time seen by the reader when reading the video frame (using time synchronization within the cluster, like NTP).

- *Historical read throughput*: This metric, measured in MBps, accounts for the throughput that a reader (or multiple readers) can achieve when reading video streams that are not cached in Pravega. As Pravega automatically moves data to a long-term storage tier, this metric illustrate the historical read performance that we can achieve when fetching video data from long-term storage. This is relevant if we consider applications using historical video streams for AI training purposes.

Experiment 1: Performance impact of bitrate.

Goal: Understand the impact of video bitrate on the IO performance of Pravega.

Observations: For the simplest scenario of 1 writer/1 reader in a video stream (see Fig. 21), the impact of bitrate for typical values (5Mbps-20Mbps) is limited. For instance, at the p50 (median), the end-to-end latency for a video stream with bitrate= 5Mbps is $<$ 1ms lower than for a video stream with a bitrate= 20Mbps. For higher percentiles, the difference seems to be even smaller and all of them show long distribution tails, typical of streaming systems.

Conclusion: NCT can efficiently use Pravega to ingest and serve video streams for a wide range of bitrate values. This is important to be able to address heterogeneous video analytics use cases.

Experiment 2: IO Performance based on the number of writers

Goal: Assess the impact on end-to-end latency of multiple parallel readers on a video stream.

Observations: As visible in Fig. 22, Pravega can handle multiple parallel video readers for a given stream. For instance, for a bitrate= 5Mbps, the p95 end-to-end latency is around 2.5ms higher when 8 readers are consuming data from a stream compared to the case of 1 reader. In our understanding, the increase of latency as we add readers is not only related to Pravega, but also to the limited resources we have for deploying parallel pods perform video IO. Recall that, in general, we only use 2 Kubernetes nodes to deploy benchmark pods in order avoid co-locating writers/readers with the Pravega Segment Store, as this would hide the impact of network latency in our experiments. Note that this is a "tail read" scenario in which video readers consume video frames as soon as they arrive. In Pravega, video frames (or, more generally, events) are written to durable storage and then cached in memory for serving tail reads with low latency. These experiments show that the Pravega cache

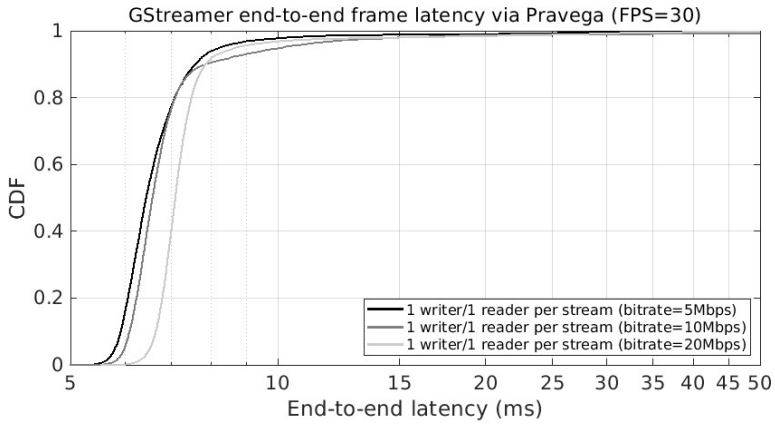


Figure 21: Impact of video bitrate on end-to-end latency.

works well for video streams and can handle multiple parallel (tail) video readers, for the bitrate values used.

Conclusion: The Pravega in-memory cache deals well with multiple parallel readers on the same video stream. This is important for NCT as there may have multiple edge analytics jobs doing performing inference from the same video stream.

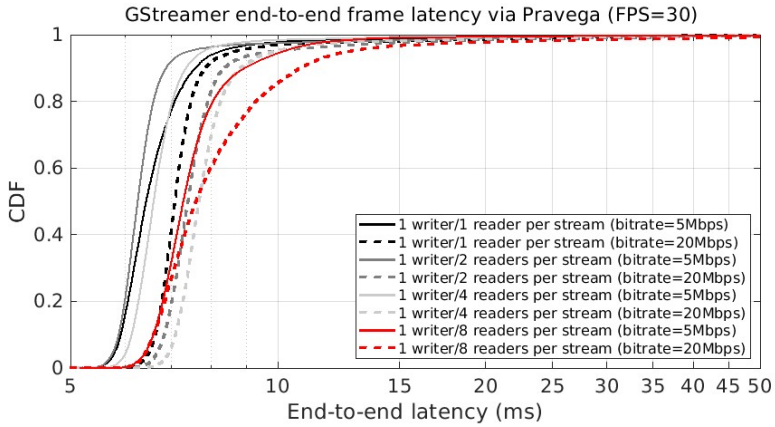


Figure 22: Impact of multiple parallel readers on end-to-end latency..

Experiment 3: IO Performance based on the number of writers.

Goal: Evaluate the impact on end-to-end latency of multiple video stream writers.

Observations: Fig. 23 (left) shows how this Pravega deployment can handle up to 4 parallel video writers (with their respective video readers) with a p95 end-to-end latency < 15ms. This is good enough for serving real-time video analytics pipelines. However, we observe that from 8 parallel writers (and readers), latency becomes too high to support real time video analytics. In this sense, we notice that the main bottleneck is not related to Pravega itself, but with the remaining resources in 2 Kubernetes nodes to run all the benchmark writers and readers. To confirm that, in Fig. 23 (right) we executed the same experiment (2 writers, 8 writers) allowing Kubernetes to use the remaining 2 Kubernetes nodes to place benchmark pods, thus having 4 nodes in total. If we inspect the results, the end-to-end latency is similar for 2 writers as there is no CPU resource issues, whereas becomes much better for 8 writers, especially for the tail of the latency distribution. This means that video analytics performance at the Edge may be severely impacted by resource availability (CPU, memory, networking) and end-to-end latency may be impacted as a side effect.

Conclusion: Even in a small deployment, NCT could exploit Pravega to efficiently handle multi-

ple parallel video writers (*e.g.*, up to 4 writers/readers with a bitrate=5Mbps the end-to-end latency at p95 is < 10ms). However, in addition to IO resources, having multiple parallel processes writing and reading from Pravega also requires from other resources (*e.g.*, CPU, memory), which may be scarce in an Edge deployment and could severely impact end-to-end latency.

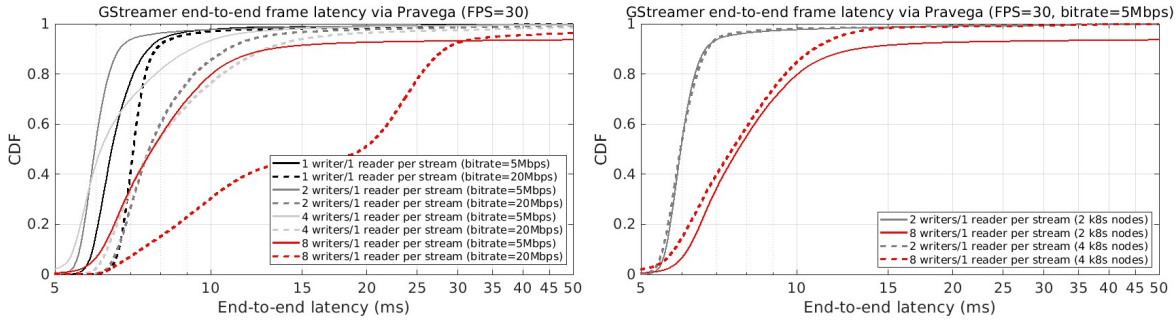


Figure 23: Impact of multiple parallel writers on end-to-end latency (left). Impact of compute resources on end-to-end latency with multiple writers (right).

Experiment 4: Impact of infrastructure on IO performance.

Goal: Evaluate the impact of networking and storage configurations on the IO performance of Pravega for video streams.

Observations: In Fig. 24, we show the impact of various storage configurations and deployment layouts on IO performance for Pravega video streams. First, the use of local drives provides an important performance advantage (2x end-to-end latency improvement) compared to remote volumes. That is, local drives provide the best possible performance to durably store data in a write-ahead log manner (each write needs to be *fsynced* for durability). In this sense, we compared the local volumes in our Cork on-premises cluster versus a similar deployment on AWS Elastic Kubernetes Service (EKS) where nodes are provisioned with local NVMe drives. We used such NVMe drives to provision storage volumes for Bookkeeper, which is the main storage component of Pravega. In this sense, we notice a small end-to-end latency improvement of AWS EKS compared to our on-premises Cork cluster using local drives, which is probably due to the type of drives used. Besides, we wanted to inspect the impact of networking on video frames end-to-end latency. In this sense, using remote storage in our Cork cluster, we evaluated the impact on latency of having the video readers and writers in the same node as the Pravega Segment Store. As visible in Fig. 24, IO end-to-end latency is reduced in $2-3$ms by co-locating pods within the same node and avoiding the use of virtualized networking.

Conclusion: Using local drives provides significant improvements on IO end-to-end latency, which is critical for video analytics applications like computer-assisted surgery. Minimizing the use of virtualized networking in video analytics applications may also provide latency improvements.

Experiment 5: Historical video read throughput.

Goal: Inspect the read throughput of readers catching up with historical video streams.

Observations: Fig. 25 shows an experiment in which a writer is writing to Pravega a video stream of 5Mbps for 30 minutes. The reader is not started until 25 minutes seconds after the writer has started. Once the reader is started, it reads as fast as possible to catch up with the writer. As visible, once the reader is started it reads up to $+90$MBps, thus catching up with the available video data in few seconds. Note that at least part of the data requested by the reader is not anymore in the Segment Store cache and should be fetched from long-term storage. The excellent storage tiering capabilities of Pravega provides high performance also in historical reads.

Conclusion: Pravega can properly serve historical video reads with high throughput. This is very valuable, as there are use cases like NCT in which historical video needs to be reprocessed in batch for different purposes, such as training AI algorithms.

Experiment 6: IO latency vs AI inference latency.

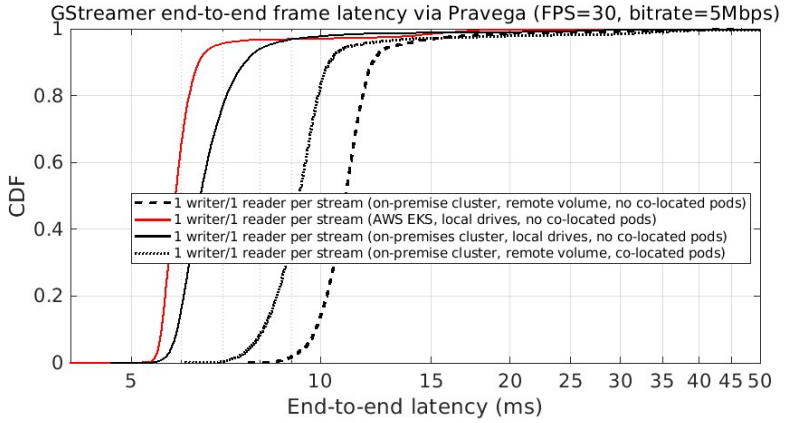


Figure 24: End-to-end latency of video frames when using local volumes, remote volumes, and benchmark pods co-located with Segment Store pod.

Figure 25: Historical throughput for a single video reader (bitrate= 5Mbps).

Goal: Understand the relative impact of Pravega IO latency on a video inference pipeline.

Observations: In Fig. 26, we show one of the AI inference jobs (see Fig. 19) used to evaluate the inference latency compared to the IO end-to-end latency. Concretely, we use two inference jobs in this experiment: i) a liver segmentation job, and ii) and surgery instrument identification job. The video inference jobs are deployed on a separate Kubernetes node with access to a GPU. In Fig. 26, we can compare the latency values related to IO and video inference. Visibly, while the distribution of inference latency is not heavy tailed (*i.e.*, GPU has a more stable behavior than network/drive), most video inference latency values are significantly higher than IO latency. For instance, at p90, inference latency is around 2x higher than IO end-to-end latency for local drives (AWS EKS, on-premises cluster). This means that ingesting video data via Pravega is affordable, given all the data durability and management benefits it provides. Interestingly, we observe that running a video inference job without a GPU is virtually impractical, due to the high latency that may result from running inference on a regular vCPU. This is an interesting hardware aspect to be considered when deploying Pravega at the Edge, if a customer needs multiple video inferencing jobs to be running in parallel.

Conclusion: With a proper storage layout, the video frame end-to-end IO latency represents only a fraction of than the actual inference latency in the NCT models tested (*e.g.*, end-to-end IO latency is 45% lower than inference latency at p95). Also, it is key to have GPU hardware available (and/or sharable) for AI inference pipelines, as otherwise the inference latency becomes impractical.

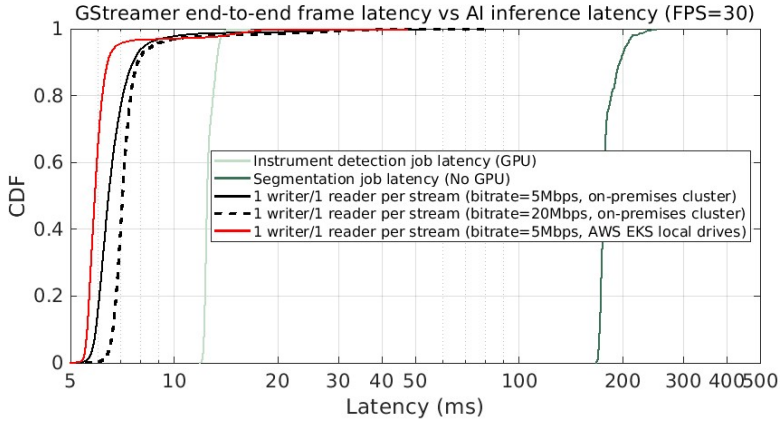


Figure 26: Latency comparison of video frame IO vs inference latency.

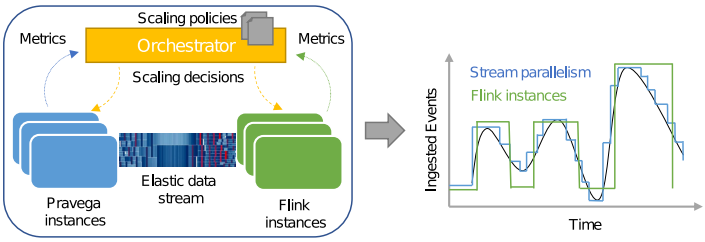


Figure 27: Orchestrator component and its interactions with user-defined scaling policies, Pravega, and Flink. Moreover, we show an example of exploiting the information about the parallelism of a Pravega stream for right-sizing a Flink job.

KPI-3: Demonstrated resource auto-scaling for batch and stream data processing, validated thanks to data-driven orchestration of massive workflows.

There are multiple angles in which resource auto-scaling can be exploited to the benefit of use-cases like computer-assisted surgery. During the first part of the project, we explored the auto-scaling of the stream abstraction itself. While a full description of this piece of research work is described in D3.1, we briefly overview the main outcomes in this deliverable and the main benefits for NCT.

Stream processing pipelines need to handle workload fluctuations (*e.g.*, daily patterns, popularity spikes) by scaling up/down the resources contributed to running jobs. This is likely the case in NEARDATA use cases; *e.g.*, surgery rooms occupancy (NCT for computer-assisted surgery) or even genomic data generation (UKHSA for genomics data storage and analytics) may show strong daily patterns due to standard work shifts. While there have been efforts proposing auto-scaling mechanisms for stream processing engines, prior work has overlooked the role of the storage system in ingesting and serving stream data. The absence of effective scaling for data streams is problematic given that the number of parallel partitions of a data stream limits both streaming data ingestion throughput and read parallelism for downstream streaming jobs.

In NEARDATA, we propose to augment the auto-scaling notion of stream processing engines with information about the source data stream. The key novelty of our approach lies in exploiting *elastic data streams* to ingest data, which is a unique feature of Pravega: a storage system for data streams part of the Dell's Streaming Data Platform. Pravega streams can dynamically change their parallelism based on the ingestion workload, and such information can in turn be exploited for auto-scaling the streaming job downstream. To this end, we have developed an Apache Flink connector for Pravega, as well as an auto-scaling orchestrator that feeds on data stream metrics (see Fig. 27). The orchestrator is in charge of consuming monitoring information from both storage and processing

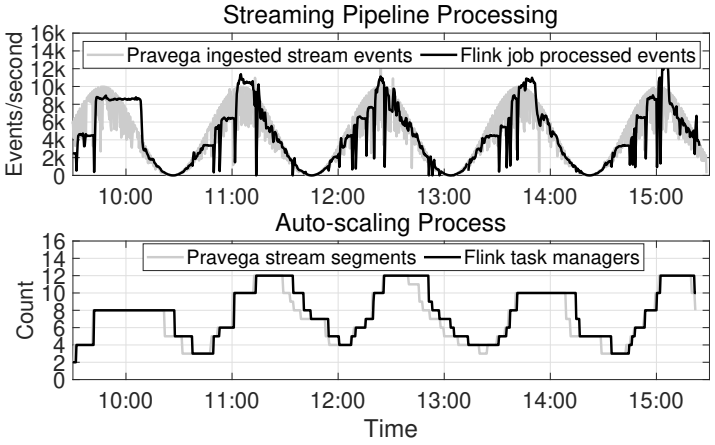


Figure 28: Data ingestion and processing overview of the streaming pipeline with our storage-compute auto-scaling mechanism.

engines. This information is the foundation for making auto-scaling decisions. Second, the orchestrator accepts user-defined scaling policies based on monitoring metrics. It continuously evaluates the metrics related to the existing policies for reacting and scaling up or down a specific service accordingly.

Experiment 1: Stream and compute coordinated parallelism auto-scaling.

Goal: We aim to evaluate the behavior of our storage-compute auto-scaling mechanism for streaming processing pipelines.

Observations: Fig. 28 shows a 6-hour experiment in which a fluctuating ingestion workload influences both the Pravega stream parallelism and the number of Flink task managers. Visibly, Fig. 28 shows how the number of stream segments changes dynamically according to the workload intensity. The scaling policy for the stream is set to 0.8K events/second. Note that during stream scaling, the Pravega control plane still guarantees event ordering per routing key as it changes the segment configuration of the stream. On the reader side, we also observe that the number of Flink task managers auto-scales according to the number of segments in the stream. The job scales because we have set a one-to-one relationship between task managers and segments in the orchestrator policy. As expected, the auto-scaling process for the Flink job has a direct effect on its processing throughput. Furthermore, despite the dynamic changes in stream parallelism, our connector and the underlying reader group semantics guarantee that all events are read only once.

Conclusion: The orchestrator component allows us to auto-scaling streaming analytics jobs automatically based on elastic Pravega stream parallelism. This may be important in use cases like NCT, in which, in addition to video cameras, there could be multiple medical devices publishing events that require streaming storage and processing.

KPI-5: Demonstrated simplicity and productivity of the software platform, validated with real user communities in International Health Data Spaces.

Reduction in **developer hours/cycles** required for data management tasks related to surgery videos, comparing the use of Pravega and GStreamer to current methods. It is expected that the storage and accessing of surgical data is significantly simplified by using Pravega and GStreamer. Our research will evaluate the simplicity and productivity of the software platform by engaging real user communities within International Health Data Spaces.

A prominent application where Pravega and GStreamer can push the usability forward is data acquisition in the surgical domain. Currently, each department utilizes individual PCs to save laparoscopic videos, followed by various data processing steps to extract essential information. Post-acquisition, data consolidation involves either laborious copying and pasting across multiple com-

puters via remote access or onto physical hard drives, both inefficient methods. Leveraging the Pravega streaming storage system significantly reduces administrative overhead. Pravega employs a tiered storage system, streamlining the data flow process.

Experiment: Comparison of time-demands between ROS and Pravega-Gstreamer

Goal: We aim to evaluate how much time may be saved by employing the Pravega-Gstreamer solution in our workflows

Observations: The full workflow of utilising such systems can be considered to have three elements: Prototyping, Deployment, Data storage and access. To compare the amount of time used in developing prototypes consisting of surgical assistance methods of interest, it is necessary to measure the typical amount of time required for these three workflow phases. ROS provides the baseline for the amount of time, as it is the currently utilised implementation means. While the exact time needed will vary for different use cases, we expect that the reduction in developer hours will be considerable, especially for Deployment and Data administration tasks. A visualization can be found in Figure 29.

Prototyping: Approximately four hours was required to create a ROS node implementation of an instrument detection method. Due to the similar modularity and abstraction of the two infrastructures, we estimate that a similar amount of developer hours/cycles is required for implementing such methods in Pravega.

Deployment: When it comes to running prototyped and developed systems on other workstations, since we rely on containerisation with Pravega-Gstreamer (and not for ROS), we estimate that the time for deployment will be considerably less in the former. This is because the environment must be specifically set up to support ROS1 (the first version of ROS), while deployment to docker with Pravega is trivial. We expect that deployment in the case of ROS could consume around one hour, while deployment with Pravega-Gstreamer should not take much longer than the time to download the relevant container image (e.g., roughly 30 minutes).

Data Storage and Accessing: At present, there is no established means of storing data generated in the process of surgeries at the NCT. This means that it is necessary to locate data on a surgery by surgery basis by sifting through a filesystem. In contrast, the way data is accessed with Pravega simplifies this process significantly by making the data searchable with dates and events. We estimate that finding one video of interest could take up to 30 minutes with our current means, while this time could be negligible using the Pravega look-up system.

4.6 Integrations with the architecture components of NEARDATA

The surgery use case is integrating into NEARDATA's architecture with two different components.

GStreamer and Pravega serves as cloud and edge connectors which are facilitating the connection between nodes. That is serving as the backbone of distributed systems. GStreamer and Pravega enable seamless communication and data transfer between nodes, particularly in streaming data scenarios.

Scone as a local connector facilitate connections between stages of data processing within a single node. Scone serves as a secure gateway for exchanging, processing, and managing data within the local environment, ensuring confidentiality and integrity in our pipeline.

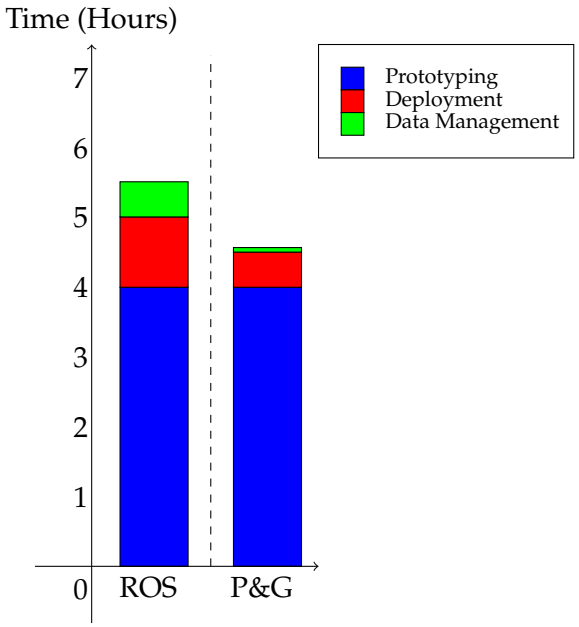


Figure 29: Pravega and GStreamer are reducing developer hours and cycles, in particular administrative workload is reduced. The graph illustrates the time required to establish a basic ROS environment for the surgical use case compared to integrating Pravega and GStreamer functionalities.

5 Transcriptomics Use Case

This use case is divided into two main experiments, which can be seen as different use cases given each present different problems and uses different datasets. Such experiments are:

- Transcriptomics Atlas
- Federated Learning for Human Genome Variation Analaysis

In the following sections we will discuss the different items separately by each scenario.

5.1 Why is the use case extreme data?

5.1.1 Transcriptomic Atlas

To build the Transcriptomics Atlas for a given cell or tissue type, the prerequisite is to obtain a set of identifiers of the input files (usually hundreds of files are needed per cell/tissue type) from the the NCBI database [9]. The input data for pipelines are *SRA* files which contain RNA sequencing data. The total size of possible subset of available SRA files that need to be processed for the atlas consist of e.g. 20 human tissues is around 10.8 TiB for 8300 files. However this is the compressed format which is not ready for processing. Those *SRA* files are converted into *FASTQ* format and increase the space requirement 7-fold. In our case the input dataset will be converted to 79.8 TiB of *FASTQ* data. The distribution of the respective *SRA* and *FASTQ* datasets are presented in Fig. 30. With the possibility of increasing the number of tissues we can easily reach order of hundreds of terabytes that requires efficient processing of such data volume.

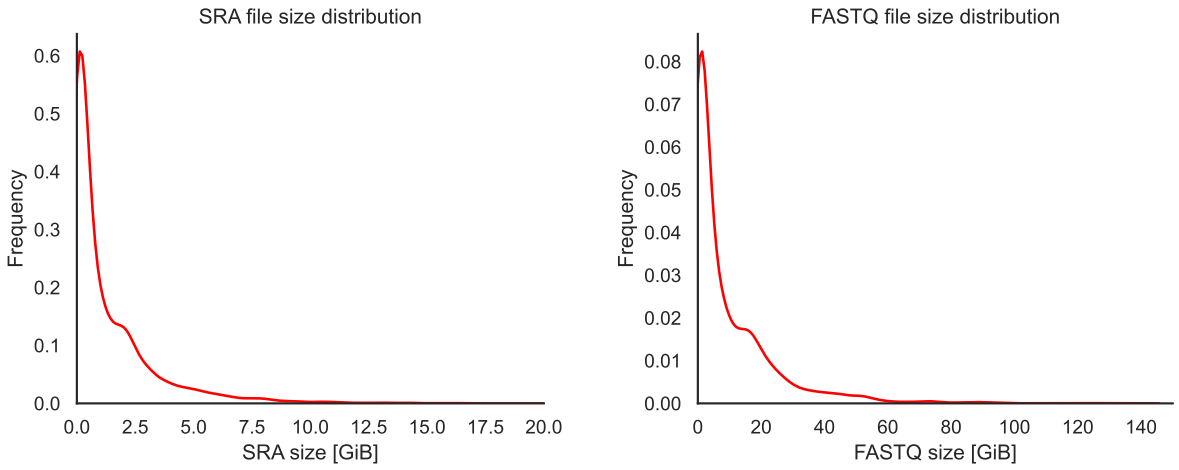


Figure 30: Distributions of SRA and FASTQ file sizes from input dataset.

5.1.2 Federated Learning for Human Genome Variation Analysis

There are several factors that make the analysis of human genome variation datasets an extreme data use case, as shown in the following sections.

Massive volume

Existing human genome variation datasets already consist of hundreds of thousands of samples, totalling at petabytes of data. This volume is still growing and will grow ever faster with the advent of new, high-throughput data collection methods. For reference, here are just a few examples of existing vast variation databases:

- Genome Aggregation Database (GnomAD): An aggregation of exome and genome sequencing data based on several large-scale sequencing projects. GnomAD currently consists of data collected from 807162 individuals, which gives a total of 897 terabytes (TB) of data [10, 11].
- 1000 Genomes project: a comprehensive repository of human genetic variation data. Data from 2500 individuals (26 populations) contributed in the project [12].
- Exome Aggregation Consortium (ExAC): the consortium collected exome sequencing data from 60000 individuals [13].
- The Cancer Genome Atlas (TCGA): a database of over 30000 tumor samples across various cancer types. The database consists of various data sources: whole genome sequencing, RNA sequencing, methylation data. The stored data volume is estimated at over 2.5 PB [14].

This is not a complete list, and new databases, that contribute to the extreme volume of human variation data are still emerging and will emerge in the future. The expanding data volume will require extreme-scale storing, retrieval and analysis solutions.

High complexity

Human genome variation data is not uniform. It usually consists of a set of several modalities, such as clinical information, DNA sequences, annotations and even spatial data (spatial transcriptomics). Moreover, having multidimensional relationships between the modalities vastly increases the complexity and presents new challenges for data analysis algorithms.

Geographic distribution

Globally collected samples are inherently distributed geographically, which presents non-trivial challenges. Transferring massive datasets to centralized location could prove difficult not only in light of the network limitations. Sensitive information encapsuled by genomic variation datasets are subject of region-specific restrictive privacy protection regulations, such as GDPR [15] and HIPAA [16]. Adhering to these regulations could prove impossible for traditional centralized data analysis solutions, which might direct data analysis solution architects to turn to near-data processing methods.

5.2 Assessed Datasets

5.2.1 Transcriptomic Atlas

Data obtained from the NCBI Sequence Reads Archive repository [17] will be used to build the Transcriptomics Atlas. We select nucleotide sequence data generated by human samples sequencing, based on tissue/cell, disease or lack thereof, and appropriate technical parameters of sequencing. The available dataset size exceeds 30 petabytes of sequence data [18] [19]. An possible alternative archive that also provides sequencing data is European Nucleotide Archive (ENA) and the volume exceeds 50 petabytes [20] [21] ENA is a repository managed by the European Bioinformatics Institute (EMBL-EBI) in the United Kingdom. Similar to NCBI SRA, ENA stores large amounts of nucleotide sequence data collected from various sequencing projects worldwide. Both organizations are part of a larger structure, the International Nucleotide Sequence Database Collaboration (INSDC), and most nucleotide data is synchronized between them. However, ENA not only contains nucleotide sequence data but also protein sequence data and genome sequence-related data.

5.2.2 Federated Learning for Human Genome Variation Analysis

In our FL experiments, we will experiment with simulated and public datasets to evaluate Federated Learning (FL) on human variation data. We will generate a substantial dataset based on the sim1000G tool [22]. It is an R package [23], specifically designed to generate human genetic variation. The software is based on the aforementioned 1000 Genomes Project database. Sim1000G takes into account several phenomena when generating datasets, including:

- Allele Frequency Diversity: the natural variation in allele frequencies observed in real human populations. Alleles are alternative versions of a gene, and their frequencies within a population influence traits and disease risk.
- Linkage Disequilibrium (LD): A phenomenon where adjacent genes in a chromosome have a tendency to be inherited together.
- Subtle population differences: Naturally occuring subtle variations in LD and allele frequency enhance the realism of generated data.

Using this tool, we can generate a realistic cohort of virtual patients and experiment with the FL workflow. In addition, we will validate the trained federation using publicly available datasets. This dual approach will give our experiments several advantages. The experiment can first be executed using a controlled dataset generated with sim1000G. We can experiment with different data characteristics to quickly assess the robustness and efficiency of the FL workflow and iterate faster. On the other hand, the public datasets, offering real-world data, give our solution a real-world relevance. By incorporating it into our experiments, we can ensure that the solution is generalizable and applicable beyond simulations. As the data we intend to use is either artificially generated or publicly available, we don't compromise privacy of the underlying patient.

5.3 Description of the use-case

5.3.1 Transcriptomic Atlas

Original pipeline was created for HPC execution by a domain specialist. It consists of four steps listed below:

1. Downloading .sra file using *prefetch* tool.
2. Converting into .fastq files using *fasterq-dump* tool.
3. Alignement and quantification of reads using *Salmon* [24] or *STAR* [25] [26].
4. Count normalization using *DESeq2* [27].

Both prefetch and fasterq-dump are provided by *SRA-Toolkit* [28]. The general pipeline is presented in Fig. 31 with alignment (STAR) and pseudoalignement (Salmon) paths respectively.

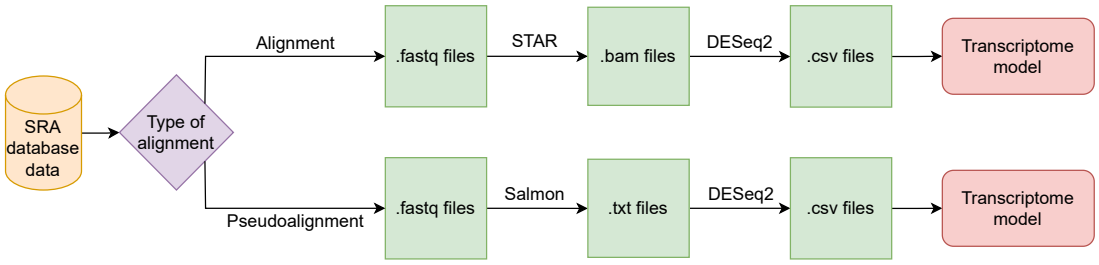


Figure 31: Transcriptomics Atlas pipeline.

In order to create a high-throughput solution in the cloud we designed and implemented architecture presented at Fig. 32. The AutoScalingGroup is responsible for sufficient number launching EC2 instances which are the workers running Transcriptomics Atlas pipeline. The selected SRA_ids are sent to the SQS queue which all the workers poll from. If there are no more tasks in the queue the instances are terminated and the desired number of instances in the Group is decreased. The results of the pipeline are sent to S3 bucket for storage along with logs. The metadata of the execution such as timestamps, file sizes, instances type etc. is sent to DynamoDB table for future analysis.

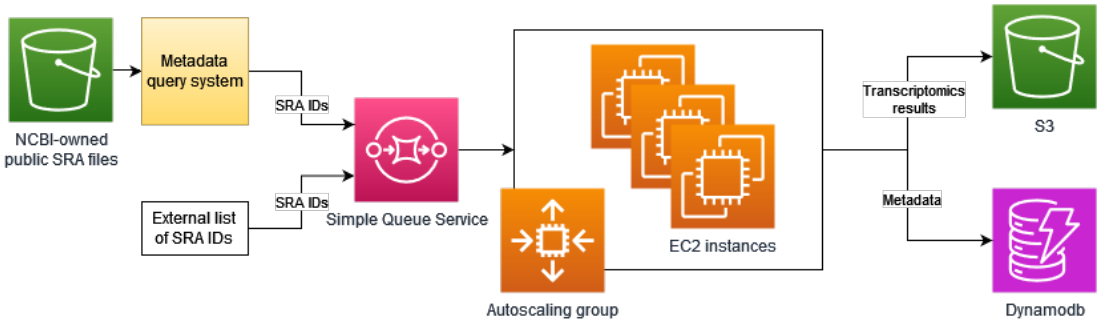


Figure 32: Cloud architecture for Transcriptomics Atlas pipeline

5.3.2 Federated Learning for Human Genome Variation Analysis

The aspects that make human genome variation data extreme-scale increase the difficulty of processing of such data with machine learning algorithms on many levels. From the scalability perspective, the traditional Centralized Learning (CL) approaches, where one model is trained based on the whole data collected in the space, is not suitable for extreme-scale environments, and some kind of parallelization is usually required to make the problem computationally feasible to solve. From the bandwidth efficiency perspective, massive data transfer to a centralized location only increases the complexity of model training, as robust, high-throghput infrastructure has to be maintained to execute it. As the genome variation data is distributed geographically and contains in its nature sensitive information related to specific patients, the applicability of CL in this context is low. Many concerns can be raised when it comes to compliance with data protection laws, such as the General Data Protection Regulation (GDPR) [29]. All of these aspects can be addressed, with Federated Learning (FL). In the spirit of the NEARDATA project, FL paradigm moves the computational resources to the data holders, instead of transferring the data to a central location. It consists of a federation of participants (institutions), where each member is an owner of his own local dataset. The standard FL workflow consists of local training rounds, where the members train their local models on the respective local datasets. After finishing the local training, the parameters of each local model are sent, to a central orchestrator, and aggregated, e.g. by averaging [30]. This aggregated, global model, is then broadcasted to all members. This process allows for model improvement based on all the data collected within the federation, with no actual training samples transfer (Fig. 33). This holds several advantages over the CL approach. As the data processing (in this case model training) is distributed, it is scalable and can handle extreme-scale datasets with the right number of members. The bandwidth required is much lower than in the CL approach, as the only data transferred within this paradigm are model parameters, which are inherently smaller in size and less complex than training data. Last, but not least is the increased privacy of the system: as there is no actual data samples transfer, the risk of leaking private, sensitive information is much lower compared to CL, which prevents non-compliance with legal regulations. To show the features of FL in the context of human genome variation data analysis, we implemented an FL framework, based on Flower [31]. The framework is written in Python, which makes it suitable for fast prototyping and can use several execution environments. We implemented Docker, SLURM and Kubernetes integration, which makes it suitable both for execution in both local clusters, HPC and Cloud environments. The data ingestion for the existing experiments is implemented using Pytorch datasets [32]. When it comes to machine learning models, the framework in general is model agnostic, which means that the FL optimization algorithm and parameters we use are invariant to model and dataset changes. This allows the framework to be used also for other experiments, beyond human variation analysis. Currently, we validated both the learning algorithm and the integration of the framework with different execution environments using standard machine learning benchmark datasets, such as MNIST [33] and CIFAR [34]. The ultimate goal is to use it to conduct human genome variation analysis on extreme-scale datasets referenced in the **??**. As the workflow is containerized using Docker and can be deployed with Kubernetes, a natural step

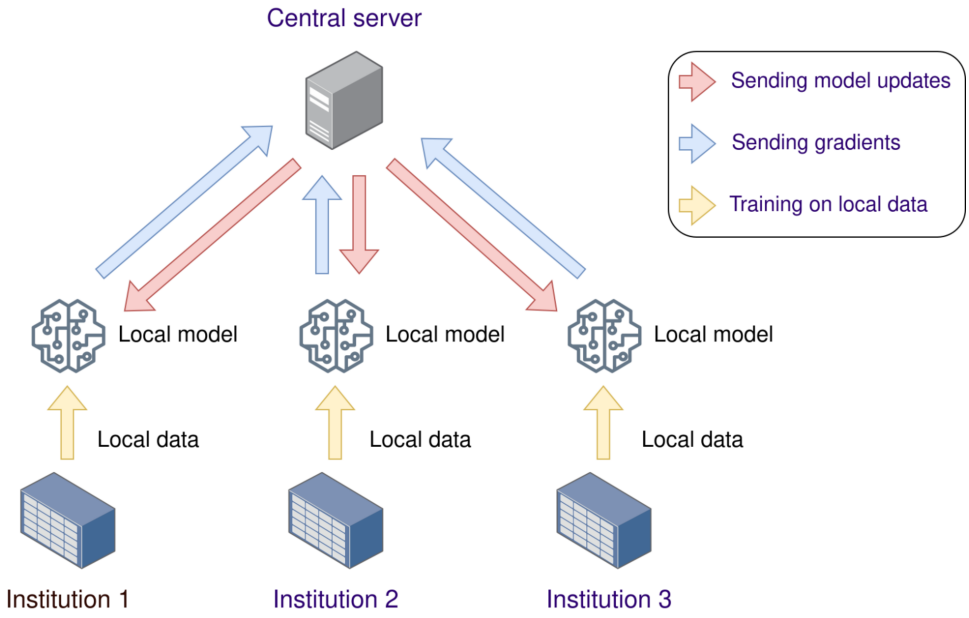


Figure 33: Diagram of the FL workflow.

to extend its security is by executing it in a Trusted Execution Environment (TEE). We plan to work closely with our technical partner SCONE [35] in that aspect.

5.4 Data Connectors

5.4.1 Transcriptomic Atlas

As less resource demanding than the other, the pseudoalignment (Salmon) pipeline has a potential to be run on a serverless architecture. Its big use of memory and storage space is caused mainly by the size of input files and resource demanding decompression mechanism. In this specific step of our pipeline, we can leverage two Data Connectors from the proejct architecture:

- **DataPlug** – for accessing fastq.gx files,
- **Lithops** – for distributing the pseudoalignment processing on AWS Cloud.

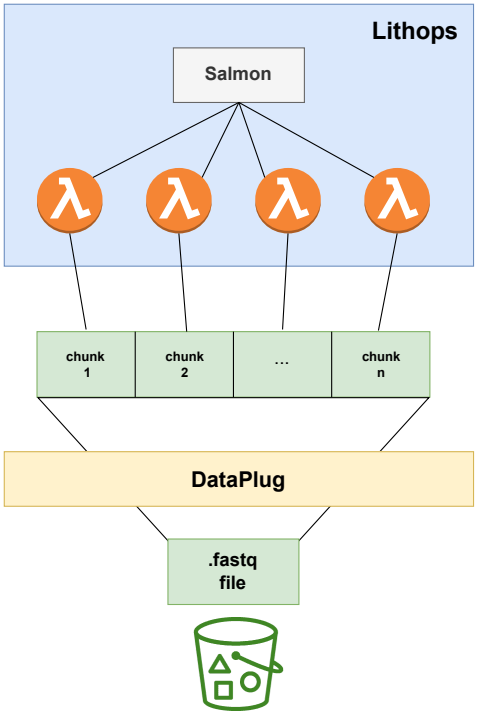


Figure 34: Parallelization of Salmon processing step using serverless processing with Lithops and DataPlug connector

Using fastq.gz format for input files and partitioning the input makes a serverless functions a viable solution. Realizing this, we provide a prototype of such pipeline with Salmon software using Lithops framework for distributing computation between lambdas. The clear choice for a data connector in this case is the DataPlug software. Gzipped fastq files need to be present on an AWS S3. The files are then logically partitioned by the DataPlug and the chunks obtained this way can be easily accesses by workers via Lithops. The scheme of a pipeline is shown in the Figure 34.

5.4.2 Federated Learning for Human Genome Variation Analysis

Federated Learning's performance, just like for any other machine learning workflow, depends heavily on the performance of data ingestion. We see the applicability of Lithops to handle parallel data processing jobs when preparing the training samples for each client over the course of the FL workflow in our experiments. Our use case is suitable for testing Lithops capabilities, as we need a data connector that can be integrated into many execution environments (HPC, cloud), and also one that can handle massive scale.

5.5 What are the benchmarks/KPI?

5.5.1 Experiment 1: Transcriptomic Atlas

KPI-1: Significant performance improvements (data throughput, data transfer reduction) in Extract-Transform (ETL) phases validated with near-data connectors over extreme data vol- umes (genomics, metabolomics).

Execution time reduction due to usage of newer release of human genome. STAR alignment step requires and index to be loaded into memory. Such index is generated on human genome available on e.g. Ensembl [36]. Index generation is a one-time task and then it can be distributed to workers in the initialization phase. By using newer release (with smaller number of patches) of the genome we generate much smaller STAR index: 28.5GiB instead of 90GiB. This allows to reduce data transfer cost and time during 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 smaller index - over 12 times faster on average (weighted by FASTQ size). We performed A/B test for 49 *SRA* files and run the pipeline with different index releases. The exact speedup values are presented on Fig. 35 and execution times are on Fig. 36. This is applicable almost exclusively to the STAR pipeline.

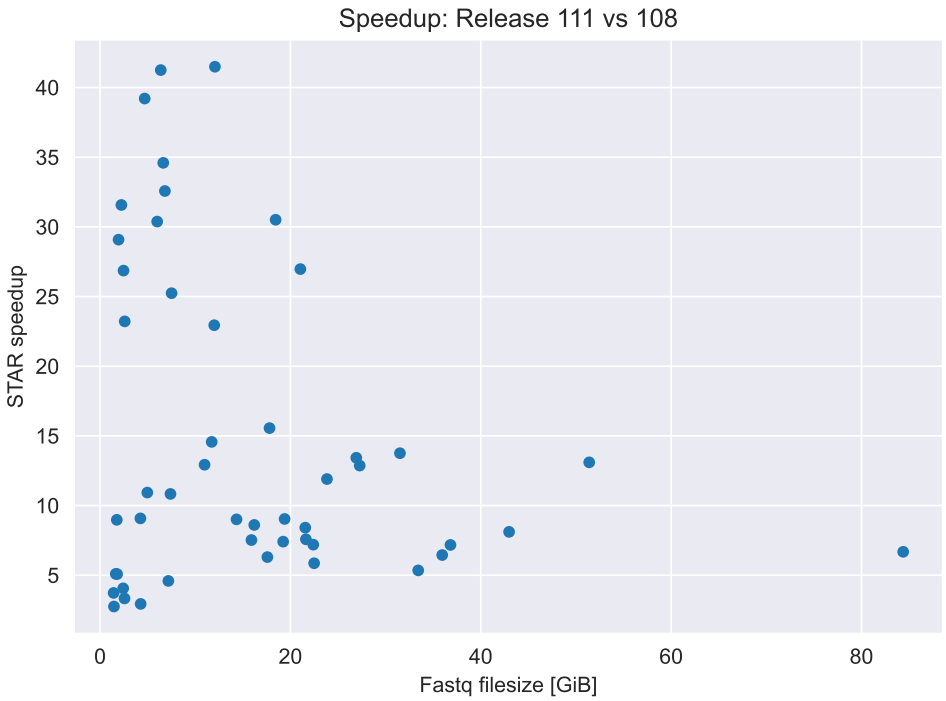


Figure 35: Acquired speedup by using index generated on Ensembl release 111 instead of release 108.

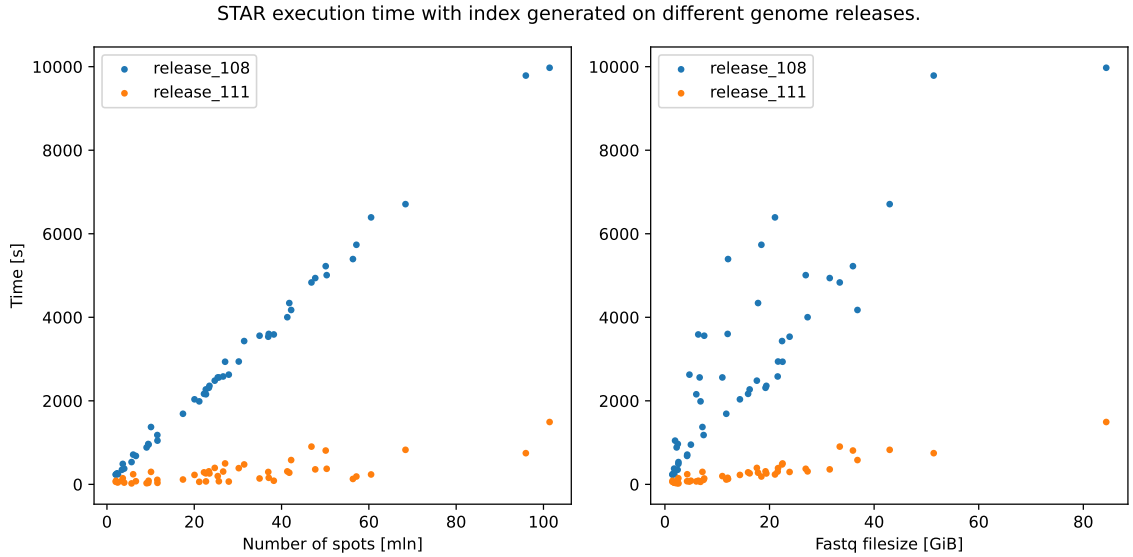


Figure 36: STAR alignment execution time using index generated on Ensembl release 111 and on release 108.

Early stopping. STAR aligner produces *Log.progress.out* file which "reports job progress statistics, such as the number of processed reads, percent of mapped reads etc. and it is updated in 1 minute intervals" [37]. Our goal is to create a comprehensive dataset of processed files for which STAR aligner returns acceptable mapping rate (above 30%). By analyzing 1000 of *Log.progress.out* files we identified that processing at least 10% of the total number of reads is enough to decide whether the alignment should be continued or it should be aborted due to insufficient mapping rate. The analysis showed that with such approach we can safely terminate early 28 (out of 1000) alignments. This would result in about 19% reduction in total STAR execution time - as shown on Fig. 37. The inputs identified for termination turned out to be single cell sequencing data, which seems to be a sub-optimal option for our task due to the lack of complete mRNA coverage within the library. Therefore, an additional filtering step was introduced to exclude this type of data early in the pipeline. This is applicable only to the STAR pipeline.

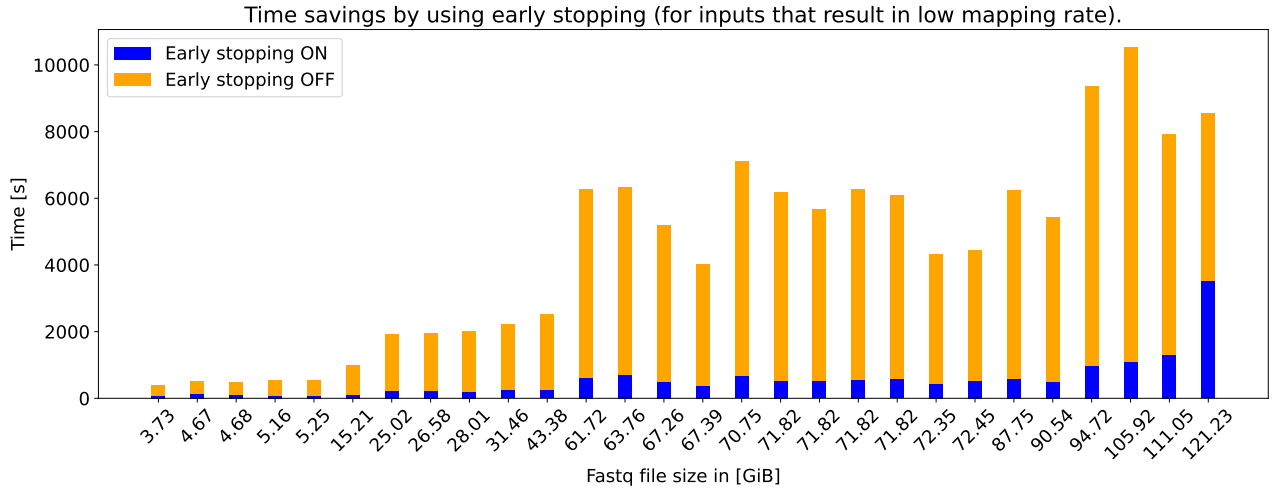


Figure 37: Possible to acquire time savings due to early stopping feature.

Serverless functions performance improvements Massive parallelism of serverless functions allows us to fully utilize potential of Salmon transcriptomics pipeline. Since the Salmon builds a statistical model from .fastq file, which consists of independent reads of biological sequences and their metadata, it easily can be broken into many independent tasks. Therefore, performing computation in distributed, asynchronous way we obtain an instant performance improvement with respect to typical use in computational biology, which is a big single machine scenario.

KPI-3: Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows.

Scalability The cloud offers 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, for example, 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 number of available nodes/cores, fair-share policies, queue system etc. which is not the case in the cloud. By using cloud services and its large compute/storage resources we can instantly launch multiple instances of workers therefore easily parallelize the workload without any queue waiting time (as seen in Fig. 38). Moreover, by optimizing cloud solution first we can create an optimized HPC alternative more easily. Genomics pipelines may not always be run efficiently and optimized for HPC as this often requires advanced knowledge of the system. Nonetheless the benchmark should compare optimized solution

in the cloud and in the HPC environment.

Index distribution to worker nodes The current cloud solution for STAR pipeline leverages Network File System (NFS) server in the Virtual Private Cloud (VPC) with EC2 worker nodes in the same region. As stated before, each worker needs to load the precomputed index into memory before running STAR alignment. Therefore as a part of the initialization phase each worker mounts an NFS endpoint to access the index. In our test, a single c6in.large instance was sufficient to provide the index to 50 EC2 instances in about 15 minutes but possible alternatives are still explored in order to improve scalability and reduce initialization time. An index distribution solution in HPC environment is part of the future work. Index distribution is much less concern for Salmon pipeline as such index is of 680MiB in size. In this case there is almost no downside/bottleneck to distribute it by e.g. saving the index in the virtual machine image (AMI on AWS) the workers are launched from.

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 contrary to on-demand model. Each running spot instance can be terminated with a 2 minute notice. Our initial analysis suggest meaningful cost reduction using this execution model. Such analysis takes into account 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 optimal scenario using spot instances should result in relatively stable computations as shown in Fig. 38. Here we present results of processing 1000 *SRA* files on r7a.2xlarge instances with STAR pipeline. During the experiment only five interruptions occurred and resulted in wasting less than 1% of the total running time among all instances.

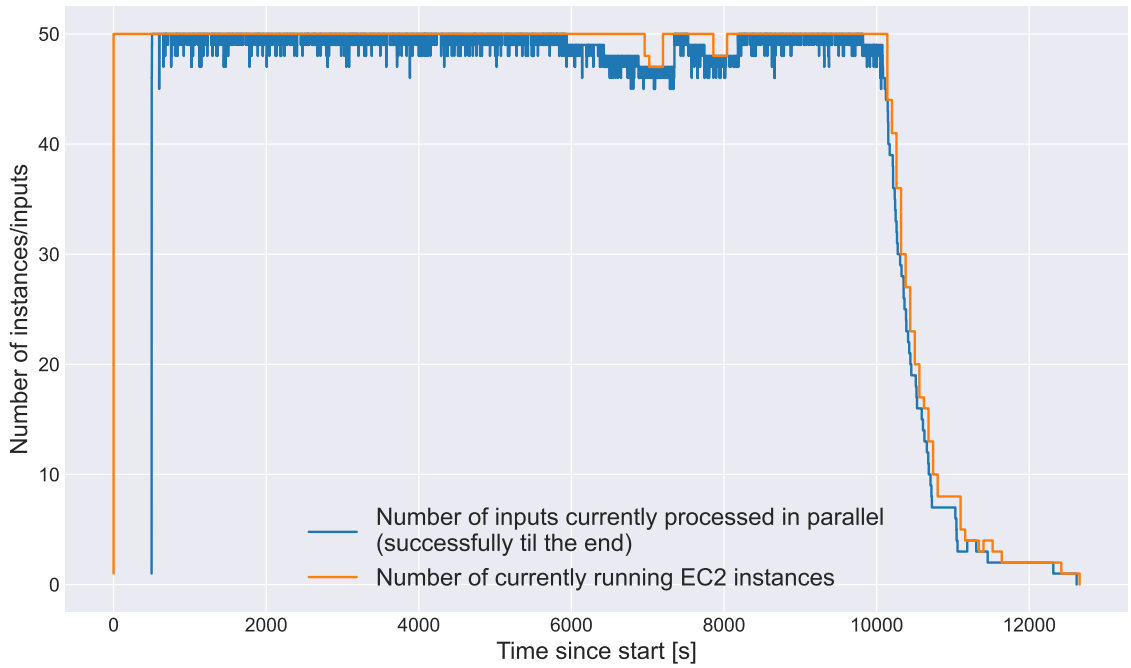


Figure 38: Spot instances usage experiment. Timeline for instances and successfully computed files.

Cost-efficient instance type for STAR There are many available instances 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 memory intensive program we decide to focus on instances with higher memory-per-CPU factor (mainly "r" family) with at least 64GiB of RAM. It is possible to use other types with 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 3. This table also presents 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 spot availability of a given type. On top of that we validate scalability of STAR. With an increasing number of threads we get faster results with good efficiency.

Table 3: 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 $

5.5.2 Experiment 2: Federated Learning for Human Genome Variation Analysis

KPI-1: Significant performance improvements (data throughput, data transfer reduction) in Extract-Transform (ETL) phases validated with near-data connectors over extreme data vol- umes (genomics, metabolomics).

The benchmark for evaluating if the Federated Learning framework proposed yields significant performance improvements for the ETL phases of our machine learning workflow consists of two experiments. The baseline setup consists of a single worker processing the whole training dataset, realising as a result the Centralized Learning approach. This is compared with a setup of N workers forming a federation, which together trains a joint model based on distributed local datasets. We expect to observe significant improvement when it comes to data throughput during training samples processing into machine learning model inputs. Moreover, we expect a significant disparity in data transfer between the two training paradigms. To this end, we performed initial experiments with a machine equiped with an eight core Intel® Core™ i7-10510U CPU and 32 Gigabytes of RAM. We compared wall-clock model input processing times (loading the data and transforming it into a Pytorch dataset) for a 4-member federation with a CL approach using benchmark CIFAR [34] dataset. The initial results suggest, that the federation of clients can perform the ETL phase of the workflow faster by 0.9375s in terms of a wall-clock time (75%), compared to a single worker realising the CL approach on the CIFAR dataset. As for the training phase, the theoretical data transfer reduction brought in by the FL paradigm can be calculated based on the model and training dataset size. Taking into account model parameters transfer, carried out during the federated workflow, the total over the course of the training (150 rounds, model size 0.237 MB) is 35.55 MB, whereas for the CL approach the total transfered data size (carried out during the setup step, before being ingested into the model) is 175.8 MB. This yields a data transfer reduction of 140.25 MB, by 80%. As the next step, we will focus on extending this benchmark, focusing on several key areas. First, we need to perform the benchmark on extreme human genome variation data volumes to see if the observations are also true in this context. Second, the abovementioned improvement in ETL phase wall-clock time stems from the distributed nature of FL. It is a perfect place to incorporate Lithops to optimize this phase even further, showcasing the true advantage of near-data processing. Third, we need to add data transfer monitoring tools, so that we don't rely on mathematical modelling for data transfer reduction calculation. Fourth, the benchmark should be performed for all execution environments compatible with our framework, to establish, how it influences the observed improvements.

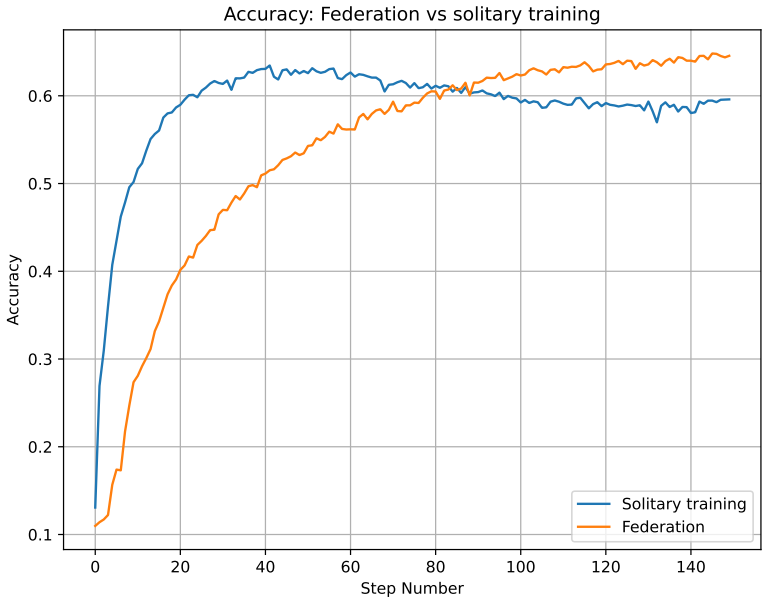


Figure 39: The accuracy improvement for a federation compared to solitary training for the CIFAR benchmark dataset.

KPI-4: High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.

Building on top of the already containerized framework, we would like to incorporate secure containers from our technical partner, SCONE [35] in the workflow. As part of the future work, we will design a set of adversarial security experiments to investigate if the composition of data transfer reduction provided by the FL paradigm and strong security guarantees of TEEs provides a full data security.

KPI-5: Demonstrated simplicity and productivity of the software platform, validated with real user communities in International Health Data Spaces.

In our initial experiments, when comparing a federation of 4 members (described under KPI1 s ection) that train a joint model, with a solitary member training a model on his own using local data only, we can observe a significant improvement in terms of the model performance. Classification accuracy comparison for the CIFAR models [34] trained during our initial experiments is showcased on Fig. 39 and shows a an improvement (5 pp accuracy difference, 8% improvement of the original result) of the joint model trained by a federation. It is worth noting that this result is an initial result obtained without any tunning of the parameters of the federated optimization algorithm, therefore the actual difference is expected to be even more extensive. As it is clear that the FL workflow has advantage in terms of machine learning performance compared to a solitary training, we need to evaluate other aspects of the system that could help us assess the practicality of the framework in real-life applications. These aspects include:

- How the orchestration of the FL workflow should be conducted? What incentive mechanisms could we use to make joining the federation attractive to the users?
- How to solve the problem with heterogeneous compute capabilities for different institutions that might join?
- How to assert sufficient data quality for each member without raising privacy concerns?

We intend to answer these questions by close cooperation with real user communities from the health data space to evaluate the productivity of the proposed software framework.

6 Genomics Use-case

6.1 Why is the use case extreme data?

Genome sequence analysis is a compute- and data-intensive task. Current sequencing techniques generate an overwhelming volume of data, with repositories accumulating several petabytes each year[38]. Consequently, institutions struggle to meet the ever-growing demands for genomics workloads. Although High-Performance Computing (HPC) installations are commonplace in biology departments, they may fall short in providing sufficient capacity to handle peak demands, reanalyze extensive public datasets generated by large consortia, or minimize the run time needed for specific analyses. To cope with this solution, institutions opt for Cloud services that meet the needs and exceptional demands for analyzing extreme genomic data. However, traditional cluster-oriented Cloud deployments are challenging due to the complexities associated with evaluating, configuring, and deploying resources and services across a multitude of heterogeneous Cloud providers. Function-as-a-Service (FaaS) offers a compelling alternative for highly parallel data-intensive applications[39][40], with minimal configuration requirements and the ability to instantly scale up and down to zero, making it an ideal choice for less experienced Cloud users.

We present a use case for genomic sequence analysis. Our use case is based on variant calling to identify variants from reference genome sequences. We discuss that this type of data is extreme due to the extreme volume of data and extensive pipelines that require large computational resources to cope with the demands needed to process and analyze this data. The objective of this use case is to port a genomics Variant calling workflow from our HPC implementation to a serverless architecture that allows processing and analyzing extreme genomic data at large scales.

6.2 Assessed Datasets

The data used to implement the serverless architecture will be the same as those used for the HPC version. Our inputs are based on two genomic data formats, the FASTA format to represent nucleotide sequences used as reference genome and the FASTQ format to represent nucleotide sequences and their corresponding quality scores that will be compared to the reference genome. We present different sizes of open data to evaluate our variant calling pipeline:

- Small size: Trypanosome: Reference genome: TriTrypDB-67_TbruceiTREU927[1] (35MB) and Sequence Reads: SRR6052133[2] (668MB).
- Medium size: Human: Reference genome: hg19[3] (905MB) and Sequence Reads: SRR15068323[4] (1.2GB) and ERR9856489[5] (12.1 GB).
- Large size: Bos taurus: Reference genome: bos_taurus[6] (781MB) and Sequence Reads: SRR934415[7] (16.5GB).

6.3 Description of the use-case

Variant calling in genomics involves "alignment" (i.e., a string similarity search) of sequencing reads, stored as FASTQ files, to a reference genome, stored as a FASTA file. Alignment mismatches are filtered and form the basis for establishing ("calling") any mutations ("variants") in the samples analyzed. Alignment requires loading large data structures ("indices") into memory allowing for fast similarity searches in the reference genome. With indices ranging up to 50 GB for large datasets, loading a full index in a lambda function is not feasible due to resource constraints. Alternatively, the reference genome can be split into smaller chunks and align reads to all of them. Consequently,

[1] https://tritrypdb.org/tritrypdb/app/downloads/Current_Release/TbruceiTREU927/fasta/data/
[2] https://trace.ncbi.nlm.nih.gov/Traces/?view=run_browser&acc=SRR6052133&display=download
[3] https://hgdownload.cse.ucsc.edu/goldenpath/hg19/bigZips/
[4] https://trace.ncbi.nlm.nih.gov/Traces/?view=run_browser&acc=SRR15068323&display=data-access
[5] https://trace.ncbi.nlm.nih.gov/Traces/?view=run_browser&acc=ERR9856489&display=data-access
[6] https://www.ensembl.org/Bos_taurus/Info/Index
[7] https://trace.ncbi.nlm.nih.gov/Traces/?view=run_browser&acc=SRR934415&display=data-access

a reduce step is required to collect and score all alignments for each read later on. FASTQ reads can also be chunked to trivially increase parallelism, as each sequencing read is aligned to the genome independently.

The original HPC implementation involved bash scripts and CLI tools that utilized the local file system and pipe commands to pass data between processes. To transition this pipeline into a serverless architecture, we employed Python code to wrap the genomic CLI tools and introduce new logic for data partitioning and load distribution across fine-grained serverless tasks.

Porting this pipeline to a serverless architecture for scalability poses several technical challenges, including the need for effective and efficient FASTA and FASTQ file partitioning with appropriate partition sizes, dealing with blocking code due to data dependencies, and managing stateful data shuffling.

NEARDATA's serverless data analytics platform Lithops provides facilities for massively invoking functions across different cloud providers. Also, the existence of parallelization in the HPC implementation allows Lithops to fit perfectly to port this implementation to a serverless architecture.

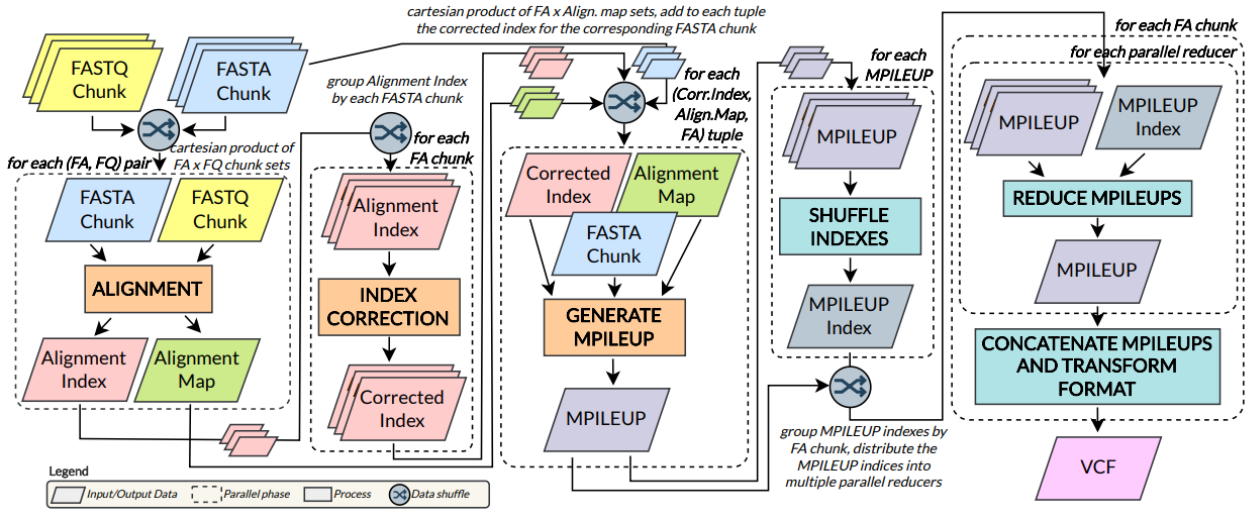


Figure 40: Big picture of the serverless Variant Calling workflow.

The serverless variant calling pipeline architecture can be seen in Figure 40. It is based on the traditional Map-Reduce model.

Pre-processing phase. This phase prepares the data so that it can be consumed in a parallel and distributed manner. Our HPC data preprocessing method is based on static partitioning. Here we encounter our first problem, the data is duplicated because the data partitions to be consumed are stored again. Also, the computational cost of partitioning is very expensive, making it difficult to study the best partition size. To solve this problem, the Dataplug Serverless Data Connector presented in NEARDATA perfectly meets the requirements of our preprocessing phase.

Map phase. This phase consists of aligning the sampled read to the reference genome. Having partitioned both, this results in a cartesian product between the FASTA and FASTQ partition sets. All FASTA× FASTQ chunk combinations are aligned using GEM3 mapper[41]. The indexed FASTA chunk is used as the reference for alignment of a given FASTQ chunk, generating an alignment file in `.map` format. Because any given alignment arises from a partial scan of the genome (a given FASTA chunk), we must add an extra step (index correction) to discard suboptimal alignments arising from the mapping of any given read to multiple genome chunks. Alignments are converted to the `mpileup` format, which collects information from every read to provide a snapshot of the cumulative alignment results for each chromosome position, providing the number of reads supporting each base call detected across all overlapping alignments.

Reduce Phase. This phase involves a shuffle of all alignment results for variant calling. To distribute the load, the map output is distributed over multiple parallel reduce functions. Data processing involves merging mpileup data across sets of functions that share the same reference FASTA chunk, i.e. those files that share the same chromosome positions. The merged mpileups are used as input for the SiNPle variant caller[42]. The partial output from each reduce function is then concatenated to produce the final output file. This can be used for any subsequent nucleotide mutation analysis workflow.

6.4 Data Connectors

As mentioned above, our serverless variant calling pipeline is divided into three different phases: Pre-processing, Map phase and Reduce phase. To facilitate the discovery and partitioning of data in the first phase and the consumption of these partitions and intermediate files generated in the other phases, we present up to four data connectors. As they are presented we will observe how each one fits into our architecture.

Data Partitioner. This connector is responsible for preprocessing the data to establish partitions that will be consumed by the processing phases. To do this, we will use the Dataplug component presented in NEARDATA. This component will allow us to avoid the problem of static partitioning and will benefit us for the evaluation of better partition sizes thanks to its dynamic partitioning based on indexes.

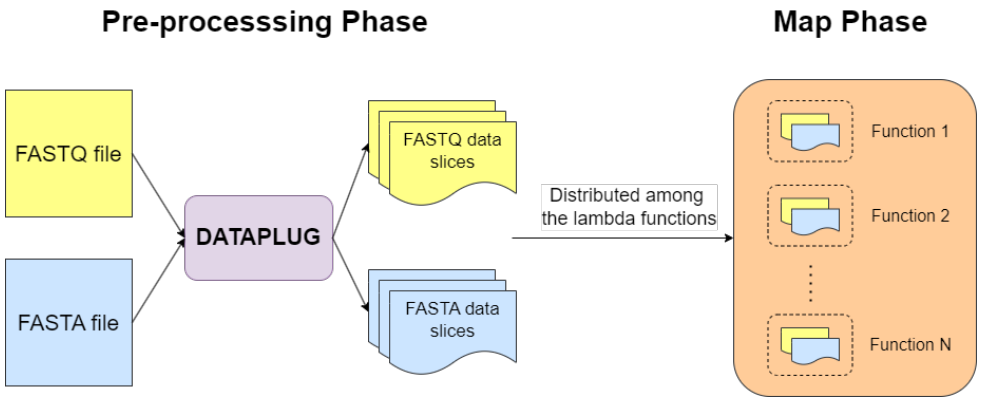


Figure 41: Architecture of the preprocessing phase using the data partitioner Dataplug.

Figure 41 shows the integration of the Dataplug serverless data connector. In this case, the combination of the two architectures fit together perfectly because we removed our previous partitioning to embed the new component. Next, instead of providing the key that referenced the static partition to the lambda function, we will pass as a parameter the data slice generated with Dataplug to be evaluated later.

Data Loader. This connector allows access to the object storage to load the generated intermediate data and to download this intermediate data for further processing in the next function phase. It should be noted that this connector will be used for the map phase. To store the objects, we will use the object storage offered by AWS known as AWS S3. In this case, we will use Lithops to access the storage backend thanks to the built-in storage API.

The Figure 42 shows how the Data Loader connector works in our architecture using the Lithops Storage API. Within the map phase, we find up to three phases of functions. The first phase Alignment, evaluates the data slices generated in the preprocessing phase, once processed the Alignment Index and Alignment Map files are generated and stored in S3. To access them, the function returns the keys with which they have been stored. In this way the Index Correction phase can access the S3 to download the Alignment Index. This same process is repeated for the following phases, i.e. each

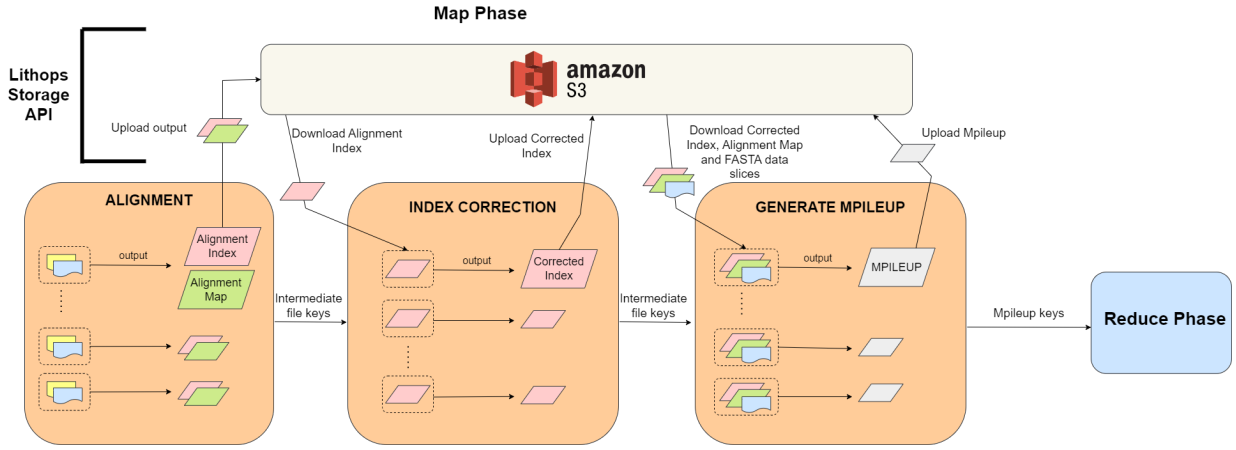


Figure 42: Map phase architecture using the Storage API and the AWS S3 service.

phase accesses the S3 to download the necessary data and finally stores the processed data again.

Data Shuffling. This data connector allows to shuffle the mpileup generated in the map phase to be distributed in the reduce phase. The complexity of the shuffling resides in the nature of the format mpileup format, which is composed of reads from the different genome positions found in FASTQ that have been with the reference FASTA. This process requires clustering the genome positions (indices) into several mpileup files belonging to the same FASTA reference fragment. However, not all indexes may exist in all files, and their number may vary. The shuffling algorithm establishes a set of index ranges that will form the partitions.

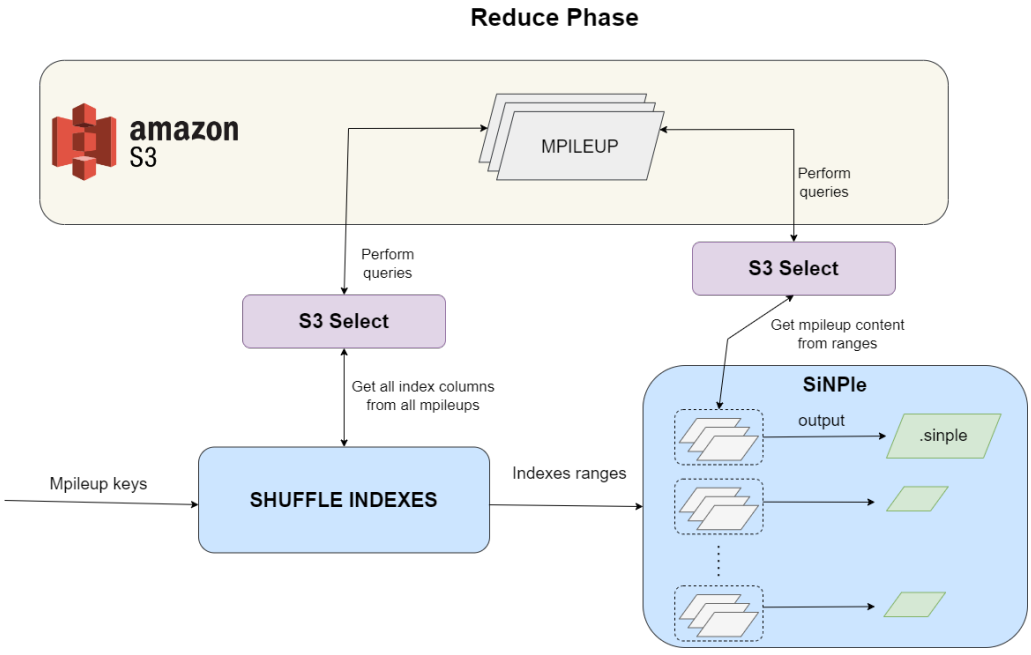


Figure 43: Architecture of the reduce phase using AWS S3 Select data shuffling.

We leverage AWS S3 SELECT for the shuffling process, as it allows to perform SQL queries over the contents of the structured text format of the mpileup files. First, we query each mpileup set to extract the column of indexes of each one of the files to be able to compute lines and to establish in a simple way the ranges of indexes for each one of the partitions. Second, each reduce function queries the file to extract directly the lines that are within the range of indexes established in the previous

phase. Figure 43 shows the integration of our serverless variant calling with the aforementioned AWS S3 Select service.

New Near-Data Shuffling. One of the biggest challenges in porting from an HPC implementation of this use case to a serverless architecture is data transfer. As we have introduced, the Data Loader and Data Shuffling connectors rely on the traditional approach of using S3 object storage to carry the data between serverless functions, consequently, this process is slow and costly, as it involves constant movement of data back and forth. Although specialized ephemeral storage solutions have been developed to address this issue, they fail to tackle the fundamental challenge of minimizing data movements.

In this use case, we present the use of the Glider[8] ephemeral storage system incorporating near-data computation to reduce the volume of data transferred for our serverless architecture. Glider aims to improve communication between serverless computing stages, allowing data to "glide" smoothly through the processing pipeline rather than bouncing between different services. Glider achieves this by leveraging near-data-state execution of complex data-bound operations and an efficient I/O streaming interface.

To address the problems described above, Glider defines **storage actions** to encapsulate computation with three key properties: (i) they are integrated as addressable storage elements in the namespace; (ii) they are defined as arbitrary objects with stateful logic; and (iii) they offer a common streaming I/O interface.

Near-data computation. Glider organizes the ephemeral storage with namespaces, a logical structure of storage elements (e.g., files or directories). Storage actions are integrated as a type of storage element. In particular, actions have a name or identifier, and applications use it to directly read or write on an action, or to organize them in the namespace. Actions are automatically managed and distributed by the system in the same way as other elements. This simplifies its usage, interaction, and management.

Arbitrary stateful code. Actions trigger a computation whenever they are accessed. This computation is defined by arbitrary, userprovided code that processes the data in and out. Thanks to being part of the storage namespace, actions are stateful and may be directly addressed multiple times. In consequence, an operation on an action may depend on the results of a previous one, which makes them great for aggregating or caching data, among many other uses. Like any other element in Glider, actions are ephemeral, as they represent intermediate data connections.

I/O streams. Consuming and producing data in small chunks is key to process large data in small-sized workers like serverless functions. Glider defines a stream-based I/O interface for all storage elements, including actions, that workers can handle with a small memory footprint. For actions, this adds extra benefits. A worker-action stream allows parallel processing at both ends in a simple and straightforward way to improve the overall performance and storage utilization (e.g., with filters or aggregations) [9].

System architecture. Glider is an extension of a multi-tiered, high-performance ephemeral storage architecture: the NodeKernel [45]. NodeKernel is a state-of-the-art storage architecture specialized on temporary data in data processing workloads. NodeKernel manages data in a set of metadata and storage servers. The metadata servers[10] administer the hierarchical namespace and the fleet of blocks. The storage servers allocate storage blocks and register them on a metadata server. The metadata servers maintain a list of free and used blocks (and their mapping to storage servers) and assign them to nodes as needed. This way, data is distributed across the cluster. To perform data operations, clients first contact a metadata server, and it replies with the location of the storage block(s) affected by the operation. The client then uses this information to perform the operation on the appropriate storage server(s). Structure operations are directly executed at the metadata server. Glider adds

[8] `https://github.com/neardata-eu/glider-store`

[9] The cloud has adopted this idea recently with a similar objective [43] [44]

[10] Metadata servers may distribute their work by partitioning the namespaces, allowing to scale the system if needed

storage actions to NodeKernel to achieve ephemeral near-data computation. A storage action is a new storage element that encapsulates a user-provided stateful computation with a stream I/O interface.

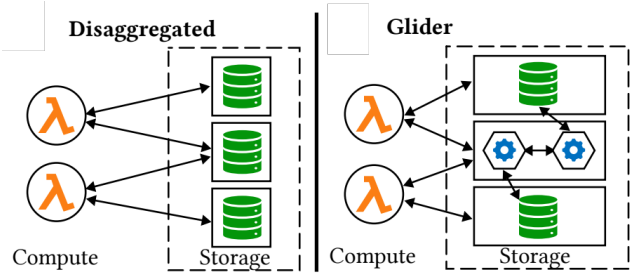


Figure 44: Architectural approaches to data-shipping.

Figure 44 shows the data lifecycle comparison of a fully serverless architecture where functions communicate through object storage with an architecture with Glider as ephemeral storage. Thanks to the storage actions offered by Glider, we can add computation close to the data. This feature fits perfectly with the Data Shuffling connector. As mentioned before, this connector is based on two phases, the first one launches an initial stage of functions to calculate the index ranges of the mpileups that will be distributed towards the next phase of functions where the contents of the mpileups will be extracted from those ranges. This procedure involves the use of the S3 Select service to connect to the object storage twice, so it involves data movements that can be reduced.

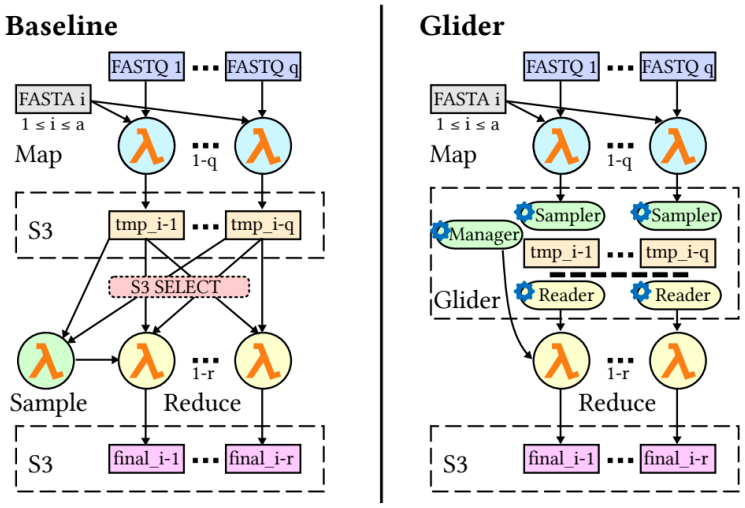


Figure 45: Diagram of the variant calling pipeline for a single FASTA chunk with serverless functions and Glider.

To address this problem, we present a new solution from Glider that can be seen in Figure 45. Instead of writing temporary files to S3 and requiring S3 SELECT to read them multiple times for shuffling, mappers directly write their output to Glider actions (Sampler Action). This first set of actions sample the data as they receive it and store it on ephemeral files. When the mapper functions finish, these actions quickly interact with a manager action that computes the number of reducers and their data ranges. Reduce functions connect with another set of actions (Reader Action). These exploit near-data computation to solve shuffling and provide reducers a single stream with the ranges of data they need from the multiple temporary files.

With this approach, similar to S3 SELECT before, reducers only ingest their range of data. However, it also adds the following benefits: (1) intermediate data is sent to a specialized ephemeral store,

faster than object storage; (2) intermediate files and actions are in the same system for improved data access; (3) a single storage request is enough to access multiple temporary files; (4) stateful computation enables more flexible data processing than S3 SELECT (limited to simple SQL SELECT queries on specific data formats); and (5) computations are streamlined with data transfers thanks to I/O streams, allowing parallelism between mappers and data sampling and reducers and data shuffling.

Data Merger. The reduction phase results in the invocation of N number of lambda functions to apply the mpileup reduction through the SiNPle program. Likewise, we find a result for each function invoked. Thanks to the correct distribution of the mpileups, we can determine the order of the results. Finally, from the established order, we take advantage of the AWS S3 Multipart upload service to perform an ordered merge of all the results obtained. Multipart upload allows you to upload a single object as a set of parts where each part will be a `.sinple` result.

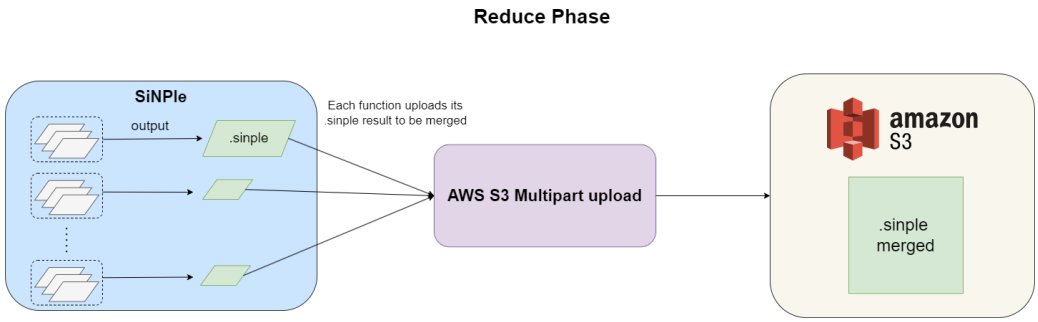


Figure 46: Architecture of the last step of serverless variant calling from the AWS S3 Multipart upload data merger.

Figure 46 shows the use of AWS S3 Multipart upload to merge the final results of all invocations of the reduce phase.

6.5 What are the benchmarks/KPI?

Next, we present the Key-Performance Indicators (KPIs) with which we validate our serverless variant calling pipeline with the incorporation of different benchmarks to evaluate the different data connectors (Glider) and integrations with the NEARDATA platform components (Lithops and Dataplug).

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).

Dealing with extreme genomic data presents a major challenge in the ETL phases. The data partitioner is the key to deal with this problem. Our initial partitioner was extended from the HPC implementation to the serverless version. Also, dragging the problem of static partitioning which leads to high computation times and data duplication. The Dataplug component of the NEARDATA platform is a perfect fit as a data partitioner. The serverless variant calling pipeline takes advantage of on-the-fly dynamic partitioning to ensure efficient preprocessing and greater flexibility in partitioning.

To evaluate the Dataplug component we present an initial benchmark for the preprocessing of the FASTA file. This bencmhark consists of analyzing the computation time of the FASTA preprocessing task. For this purpose, the implementation of the old static partitioning is compared with the solution offered by the Dataplug component.

Experiment setup. We will use the FASTA hg19 file presented above which has a total uncompressed size of 3GB. Thanks to the nature of the FASTA format, Dataplug allows us to partition it

in parallel. Also, we will set different sizes to split this file [150MB, 300MB, 450MB, 600MB] so that Dataplug can preprocess it in parallel through Lambda functions. It is worth mentioning that we will use Lithops as a computate backend, since Dataplug offers a native integration with this NEARDATA component. We will evaluate this component independently later on. Finally, we allocate a 2GB memory for the lambda functions.

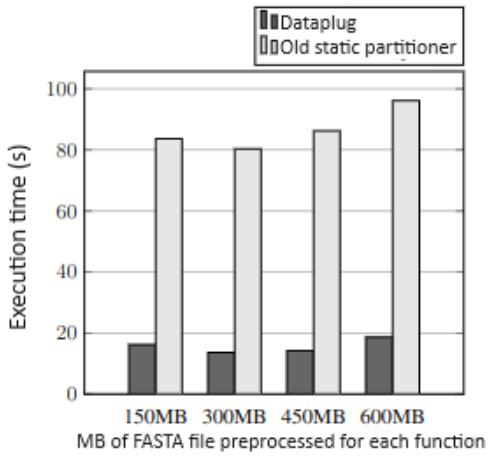


Figure 47: Dataplug performance vs Old static partitioner.

The figure 47 shows the comparison of the execution times between the old static partitioning versus the Dataplug component. As can be seen, Dataplug offers up to a 4x improvement in execution time. This type of experiments allows us to analyze the best size of the FASTA file to partition. The larger the fasta size the less functions are needed to preprocess the file. With the observed results, the execution time is similar in all cases, while the resources used are different. Consequently, using a larger size to invoke fewer lambda functions would present better results in terms of cost. This is not always the case, as the cost also depends on the memory used by the function.

For the evaluation of the FASTQ file we redirect the reader to the deliverable *D3.1 XtremeHub first release and documentation* where an extended experiment on this format is presented.

As we have been able to demonstrate, the integration of Dataplug in our variant calling pipeline guarantees improvements in the execution time in the extreme data preprocessing phase, and thanks to the new on-the-fly dynamic data partitioning, data transfers and data duplication are reduced, since it avoids uploading again the partitions to the object storage.

As we have been able to demonstrate, the integration of Dataplug into our variant call pipeline guarantees runtime improvements in the extreme data preprocessing phase, and thanks to the new dynamic data partitioning on the fly, data transfers are reduced by up to approximately 200% and data duplication is avoided, since the file is no longer completely downloaded, partitioned and uploaded again to the object storage.

KPI-3 - Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows.

Lithops as a compute backend. Lithops is the basis of our serverless architecture. The integration of this NEARDATA component provides us with the necessary tools to deploy our implementation of serverless variant call pipeline on the AWS cloud provider thus ensuring the orchestration of our massive workflow. From the previous sections, we have been able to observe the different phases of our serverless architecture based on the Map-Reduce model. Our solution presents more complex phases that require us to differentiate different stages of functions in each of the map and reduce phases. From the computate API, we can make groups of function invocations capable of executing the different function stages. Likewise, the storage API is used as a Data Loader to transfer the processed data between these stages through the use of the S3 object storage.

To validate our serverless architecture with Lithops integration, we will now evaluate evaluate two key aspects: (i) Performance evaluation where we will compare its execution time to the single-node HPC version, and (ii) Scalability evaluation where we will use a larger dataset to assess the architecture's scalability.

Experiments setup: The HPC configuration consists of a Dell PowerEdge R440 node equipped with two Intel Xeon Gold 5118 CPUs @ 2.30GHz, with a total of 24 cores (48 threads) and 32GB of memory. For the serverless AWS Lambda execution, we utilized a Docker function runtime configured with 8192 MB of memory (which provides 4 vCPUs) and an ephemeral disk size of 10240 MB.

Performance. As a first benchmark, we used the Trypanosome genome and sequence reads SRR60521335 from NCBI for the execution of the complete pipeline with single-node HPC version and Serverless version.

Table 4: Comparison of execution times between HPC and Serverless for the Variang Calling Stages

Stage	**HPC**	**Serverless**
Genome Indexing	0 min 14.20 s	0 min 9.81 s
Alignment	0 min 14.20 s	0 min 48.10 s
Index correction	-	0 min 7.63 s
Generate mpileup	51 min 15.79 s	1 min 6.55 s
Index shuffle	-	0 min 10.73 s
Variant Calling	54 min 5.04s s	0 min 27.82 s
Total	106 min 8.21 s	2 min 50.64 s

Table 4 presents execution times for the stages detailed above. The HPC version efficiently leverages multithreading for genome indexing and alignment processes but experiences performance bottlenecks in the mpileup and variant calling phases as they predominantly consist of sequential code. Conversely, the serverless setup optimally distributes workloads, resulting in a significant reduction in overall execution time even for sequential code. Thus, the serverless version is x37.46 times faster than the HPC version.

Scalability. Next, we wanted to assess the scalability of our architecture by using a larger input dataset such as the hg19 FASTA reference genome with 20 partitions and the SRR15068323 FASTQ read set with 60 partitions from the NCBI archive. This represents a realistic benchmark of variant calling in the field of human genomics. The files have an uncompressed size of 3.0 GB and 5.4 GB, respectively.

Figure 48 illustrates the pipeline's tasks execution time and concurrency over time, using 1200 tasks for the alignment stage and 112 for the reduce stage. Our asynchronous code redesign allows us to comfortably surpass AWS Lambda's 1000 concurrency limit, ensuring scalability. The FaaS platform adapts to the flexible parallelism throughout the pipeline for optimal cost-effectiveness, leaving no idle resources underutilized. The serverless architecture allowed us to scale this workload, and achieve an execution time of 7 minutes and 16.57 seconds. For reference, the same workload ran for 14 hours on the HPC setup. The total execution cost was 10.68 USD for AWS Lambda GB-sec billing, with an additional 0.0644 USD for 32.2GB data scanned in S3 SELECT.

As we have been able to demonstrate, Lithops allows us to orchestrate our massive workflow from its simple interface. This guarantees significant performance improvements over the HPC version. Additionally, the elasticity and flexibility of cloud services favor the massive parallelization of our genomics pipeline adapting perfectly to the scalability of resources needed to deal with extreme data.

Glider as ephimeral storage for near-data shuffling. In the first part, we have evaluated the serverless variant calling pipeline from the first Serverless solution that includes as Data Shuffling the AWS S3 Select service. As a new contribution at NEARDATA, the novel ephemeral storage Glider

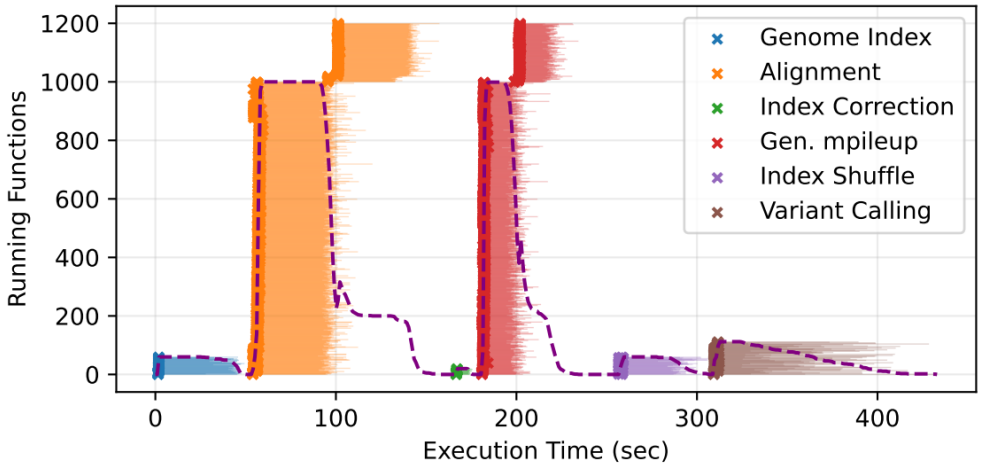


Figure 48: Taks run time and concurrency - Scalability experiment with Lithops.

was presented. In this section we will show the performance comparison between the version with S3 Select and the version with Glider. For this experiment we will focus only on the reduce phase of our pipeline.

*Experiment setup.*We run this comparison with the Human Genome FASTA file (3 GiB) and the SRR15068323 FASTQ file (5.25 GiB). These are split into $a = 20$ and $q = 35$ chunks. Hence, the full experiment runs $m = 700$ mappers. The intermediate files (about 32 GiB) are shuffled to $r = 2$ or 3 reducers per FASTA chunk (total of 47). We run partial executions with a subset of the chunks to evaluate scale. Map and reduce Lambda functions are configured with 2 GiB and 8 GiB respectively. Glider is deployed with one metadata server (t3.xlarge), up to 4 data servers (r6i.large), and up to 20 active servers (c5.12xlarge). Data and actions are uniformly distributed across nodes in their tiers.

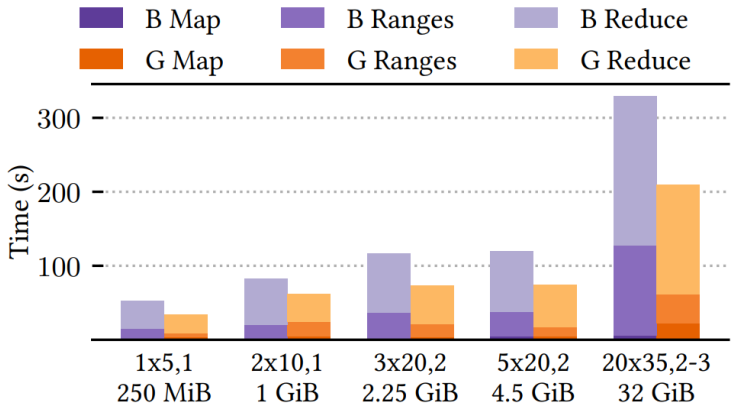


Figure 49: Serverless genomics variant calling. Execution time with Glider (G) against the baseline (B). Labels are a×q, r showing the number of chunks and reducers per FASTA chunk with an approximate size of the temporary files.

Figure 49 draws the results. Compared with the baseline, Glider allows to avoid a full read of the intermediate data to perform sampling. This minimizes storage accesses and eliminates 1/3 of the intermediate data transfer. The baseline transfers the intermediate data 3 times (mappers write, samplers read, and reducers read), while Glider only twice (mappers and reducers). Consequently, our solution with actions is always faster than the baseline. The plot shows how this behavior is kept with scale. In particular, Glider reduces execution time by 36% with the full data. The map stage is

slower with Glider since it includes data sampling at the actions. However, this allows a much faster range distribution. Finally, the reduce is also faster due to the benefits above, including lower data ingestion to workers, a richer data processing on actions, and streaming data transfer.

The incorporation of Glider as Near-Data shuffling demonstrates a reduction of the data transfer while ensuring the workflow orchestration from the distribution of the actions and the storage described in the namespace. Thanks to its modularized design, it allows the user to add data or action storage nodes according to the user's needs, thus guaranteeing the scalability of the system.

7 Metabolomics Use-case

In this use case, we contribute to improving METASPACE, a cloud platform for spatial metabolomics [11]. Spatial metabolomics is a bioanalytical technology for spatially-resolved detection of metabolites, lipids, drugs and other molecules in tissue sections used in biology, medicine, and pharmacology. Spatial metabolomics generates large datasets because for each pixel it produces a mass spectrum containing in the order of 10^4 dimensions representing abundances of different molecules. A key problem in spatial metabolomics is the metabolite identification or associating the spectral dimensions with specific metabolites they can represent. The individual spatial metabolomics datasets have large sizes, ranging from 1 to 100 GB per tissue section.

7.1 Why is the use case extreme data?

METASPACE represents a use-case of an extreme data in the health field because of the following parameters:

Volume: The METASPACE hosts over 45 TB of data. Importantly, we observed a similarity to the Moore's law with a half of this data amount submitted in the last year. This was due the fact that the latest imaging mass spectrometers are able to collect larger dataset with more pixels and more molecular dimensions. We expect this trend to continue, anticipating having over 100 TB of data by the end of this project (December 2025).

Variety: Figure 51 illustrates the variety of the datasets coming into METASPACE. Briefly, the data is being submitted by a large number of users collecting data from various samples. Moreover, the data comes from different technologies which affects the datasets size (the number of both pixels and molecular dimensions), the properties of data (noise, spatial structures), as well as the properties of the results (numbers and types of metabolites identified in data).

Veracity: Due to a wide range of users and their experiences with the spatial metabolomics technology, as well as the conscious lack of filters or QC criteria on the incoming datasets, the datasets vary highly in their quality with some datasets. The mathematical foundation of our engine [12] allows to control the False Discovery Rate in the results thus overall control the quality of the output. However, these differences in the veracity lead to varying sizes of the results thus putting a strong requirements on the scalability of the architecture handling the results.

Velocity: The numbers of datasets submitted in a day to METASPACE ranges from a few to over a hundred that requires a highly scalable computing infrastructure and the implementation. Moreover, the ranging sizes of the datasets and the selected molecular databases (sets of molecules to search for) change the volumes of data that need to be processed every day.

International Data Space: As discussed in the previous section and see from the Figure 50 highlighting the world-wide map of the users, METASPACE represents the International Data Space in the field of spatial metabolomics. This poses a number of challenges in terms of computing and offers opportunities in terms of increasing the value of this data. In this project, we are increasing the interoperability of this data (URV with EMBL) as well as molecular content of this data (in Experiment 1, to be presented later).

7.2 Assessed Datasets

In this experiment, we used two sets of datasets.

First, for the scientific evaluation and stress testing of the ML-based version of the engine, we used 1.710 datasets from 159 researchers from 47 labs encompassing both animal and plant-based datasets with a balanced representation of various spatial metabolomics contexts (the list and breakdown are available in the pilot study preprint [13]).

[11] http://metaspace2020.eu
[12] https://www.nature.com/articles/nmeth.4072
[13] https://www.biorxiv.org/content/10.1101/2023.05.29.542736v2.supplementary-material

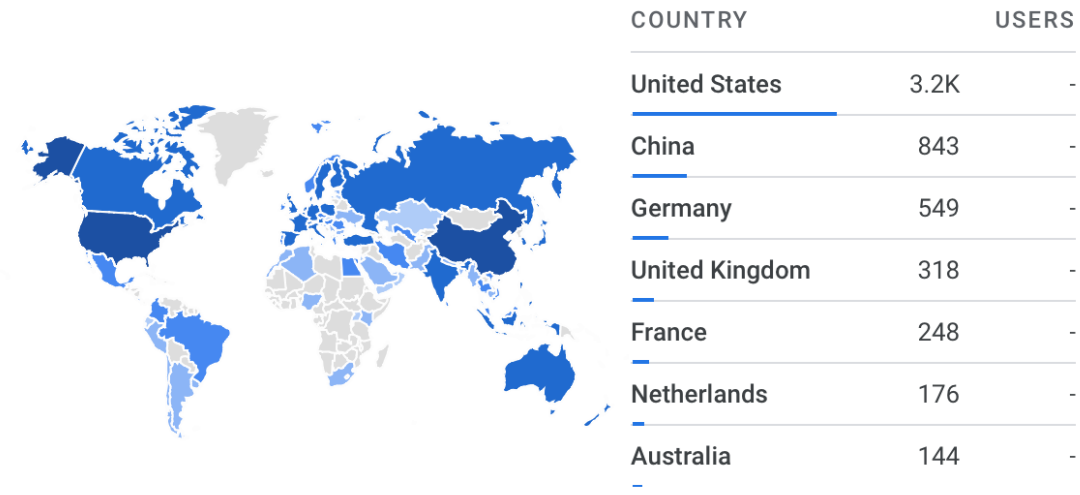


Figure 50: The map of the METASPACE users from the top-visiting countries in the last 12 months, according to the Google analytics.

Second, for the software KPIs to be presented later in this deliverable, we used 5 representative datasets of different sizes [14] processed against 4 different databases (sets of metabolites to be searched in the datasets).

Table 5: Metabolomics use case datasets used for KPIs.

Dataset ID	Dataset	Submitter	Submitter organization	Volume, in MB	Country
ds1	Brain02_Bregma142_02	Régis Lavigne	University of Rennes 1	75	France
ds2	AZ_Rat_Brains	Nicole Strittmatter	AstraZeneca	754	UK
ds3	CT26_xenograft	Nicole Strittmatter	AstraZeneca	1828	UK
ds4	Mouse brain test434x902	Dhaka Bhandari	Justus-Liebig-University Giessen	4450	Germany
ds5	X089-Mousebrain_842x603	Dhaka Bhandari	Justus-Liebig-University Giessen	7748	Germany

All software KPIs were calculated for all 5 datasets when annotating them against each of four molecular databases: PamDB, CoreMetabolome, LipidMaps, and HMDB, which among the most commonly used in METASPACE [15].

[14] https://github.com/metaspace2020/Lithops-METASPACE?tab=readme-ov-file#example-datasets
[15] https://metaspace2020.eu/help

7.3 Description of the Use-Case

METASPACE was developed in the EU Horizon2020 project METASPACE in 2015-2018 (grant ID 634402). Furthermore, in the EU Horizon 2020 project CloudButton 2019-2022 (grant ID 825184), we have developed a novel version of METASPACE underlying metabolite identification engine by employing the serverless framework Lithops [16]. The serverless version of METASPACE helped providing better service to the users and was recognized as a notable Deep Tech innovation by the EU Innovation Radar [17].

METASPACE became a leading software in the emerging field of spatial metabolomics; see Figure 51). It can be used for data from various samples and technologies with various applications in multiple fields of biology, medicine, and pharmacology. METASPACE provides capacities for big data analysis, results sharing, and can be used in conjunction with other software through API. It is used by more than 2600 users from over 160 groups from multiple contributing institutions worldwide. We have processed more than 45.000 datasets submitted, with more than 10.000 of them made public under the CC-BY 4.0 license. Most users access METASPACE through our web app; however we also provide headless access through API which is used in particular by major users.

METASPACE is open-source [18] and free supported by the funding from the EC, National Institutes of Health (NIH) from USA, as well as from a number of other grants. Importantly for the field and for this project, a large number (30%) of the datasets are made public by their submitters (under the CC-BY 4.0 license) that created the largest data space in this field.

For processing large numbers of diverse datasets, we have implemented METASPACE as a scalable data processing platform using the serverless framework Lithops, depicted in Figure 52. Moreover, the data storage infrastructure on METASPACE allows to store all the incoming files together with the processing results (identified metabolites) and other metadata. Overall, accumulated data on METASPACE amounts to over 45 TB as of April 2024.

Altogether, the scientific position and reputation, a worldwide community of users, the large number and size of datasets, and cloud architecture of data storage position METASPACE as an **International Health Data Space** in the field of spatial metabolomics.

In this project, we aim to further improve the open METASPACE cloud platform and strengthen its position as a International Health Data Space.

7.4 Data Connectors

We have developed several Data Connectors supporting the extreme data in METASPACE for ML-based metabolite identification:

- **Data submitter:** This data connector allows to submit a spatial metabolomics dataset to METASPACE to be subject to the ML-based metabolite identification.
- **Feature computer:** This data connector computes features needed for ML-based metabolite identification using Lithops
- **FDR estimator:** This data connector estimates the FDR values for all molecules from the provided molecular databases so that the molecules with FDR below a threshold can be reported to the submitter
- **Results downloader:** This data connector enables a user to download the results for either their dataset or a public dataset using its dataset ID

These data connectors allow users to efficiently perform interactions with METASPACE at the same time ensuring scalability and access to over 45+ TB of historic datasets (provided their are avilable to the user) including 10+ TB of public datasets (numbers as of April 2024).

[16] https://lithops-cloud.github.io
[17] https://innovation-radar.ec.europa.eu/innovation/50800
[18] https://github.com/metaspace2020/metaspace

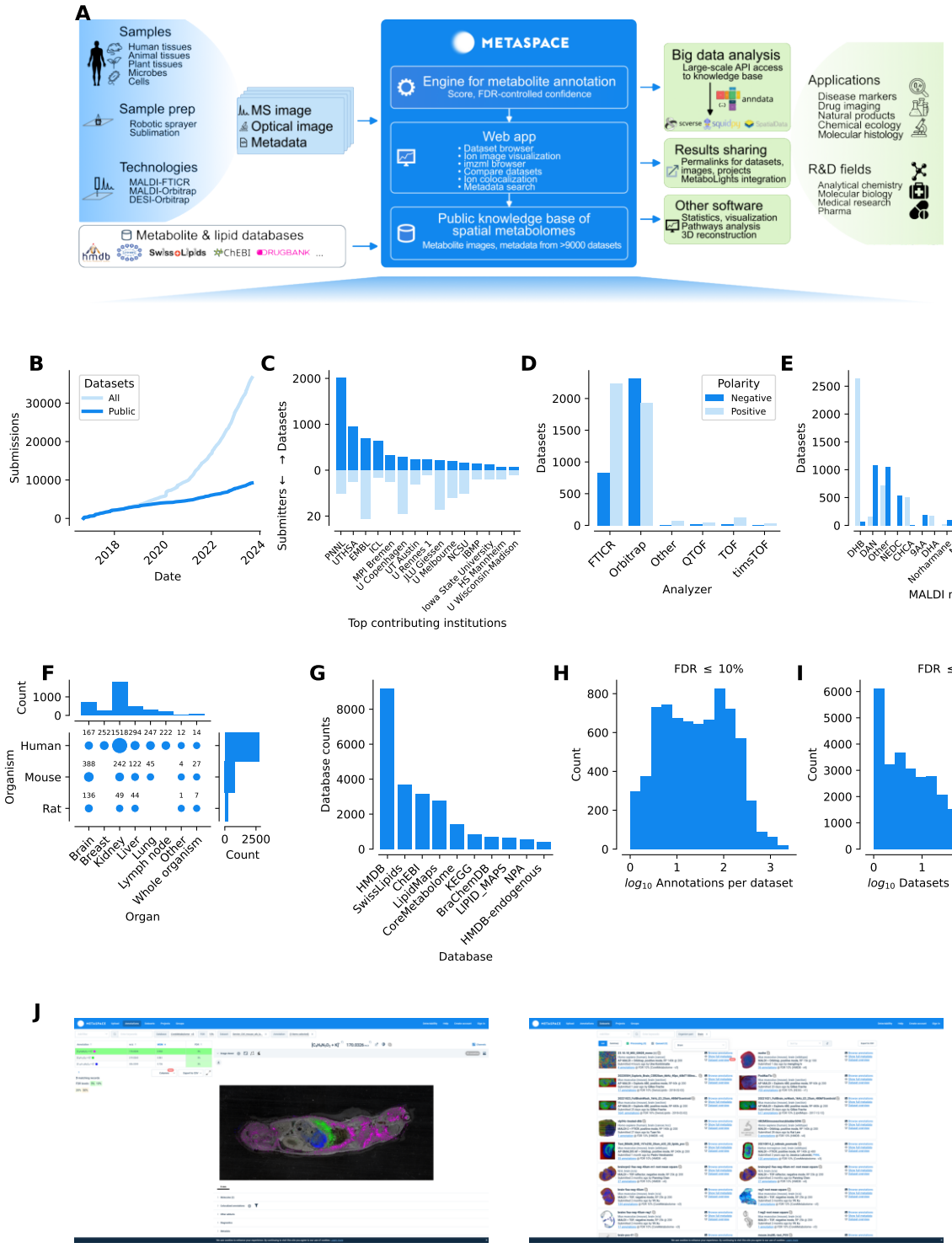


Figure 51: The overview of the METASPACE cloud platform engine and International Data Space outlining key functionality and types of usage, with key statistics of the usage illustrating the variety of the incoming datasets, and screenshots of the web app.

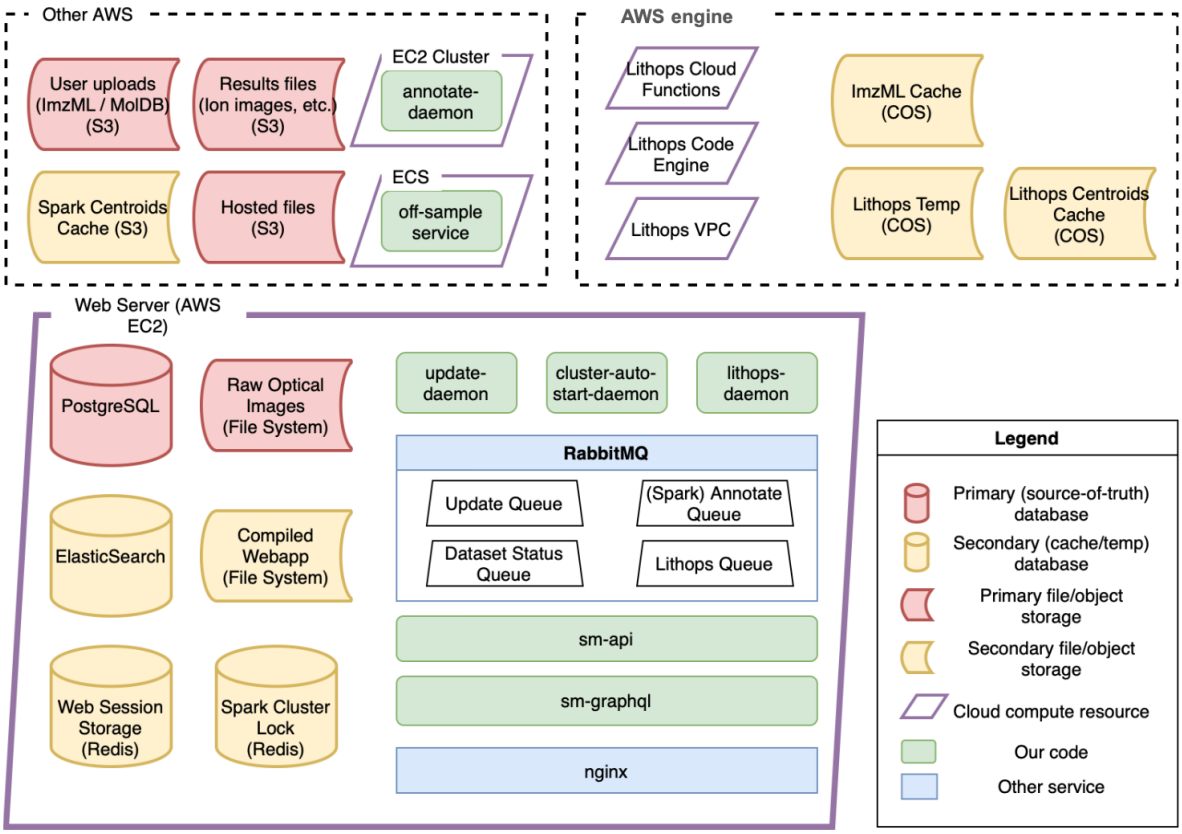


Figure 52: The architecture for scalable processing of extreme data in METASPACE by employing the serverless framework Lithops.

7.5 What are the benchmarks/KPI?

7.5.1 Experiment 1 - Mining historic METASPACE datasets with the recently developed ML model to address the "dark matter" problem in spatial metabolomics.

Here, we are describing the results and KPI benchmarks for the Experiment 1 in the metabolomics use-case, namely "performing machine-learning-based metabolite identification". In this experiment, we improve the fundamental part of the METASPACE, namely the engine for metabolite identification. The engine is implemented following our original publication in 2017 [19] where a rule-based solution was proposed based on the expert knowledge. However, this approach has its drawbacks, and at the same time it is not flexible and cannot be adapted for specific users or technologies. Addressing it, we have developed a machine learning-based approach for metabolite identification in spatial metabolomics. In our pilot study [20], adding more features and using a machine-learning based approach allowed us to significantly improve the results of data analysis, helping find molecules which could not be found with the rule-based method. In our Experiment 1, we aim to implement this new machine learning-based approach for the production version of METASPACE. Moreover, in order to support the extreme data use and the position of METASPACE as an International Data Space, we developed and optimised a number of data connectors for metabolomics including **data partitioner**, **feature analyser**, and **FDR estimator**. Implementation of this new version of the engine can increase the value of the historic and future extreme data in METASPACE for the scientists from the field of spatial metabolomics.

KPI-1: Significant performance improvements (data throughput, data transfer reduction) in Extract-Transform (ETL) phases validated with near-data connectors over extreme data volumes (ge-

[19] https://www.nature.com/articles/nmeth.4072
[20] https://www.biorxiv.org/content/10.1101/2023.05.29.542736v2

nomics, metabolomics).

In our pilot study, we have first evaluated the scientific performance of the new ML-based version of the metabolite identification by comparing it to the state of the art rule-based metabolite identification currently used on METASPACE. For this we considered the following metrics: **PR-AUC** (Precision-Recall Area Under Curve), **MAP** (Mean Average Precision), and **Number of Identified Metabolites**.

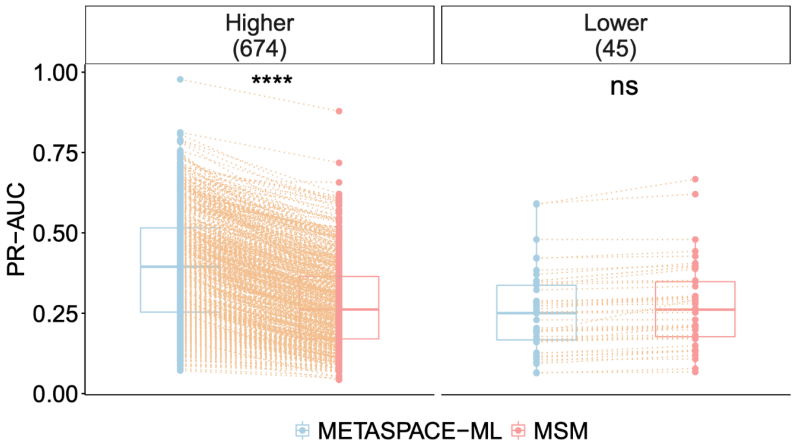


Figure 53: The PR-AUC calculated for each of the 720 testing datasets, showing an increase from 0.25 (rule-based) to 0.45 (ML), on average.

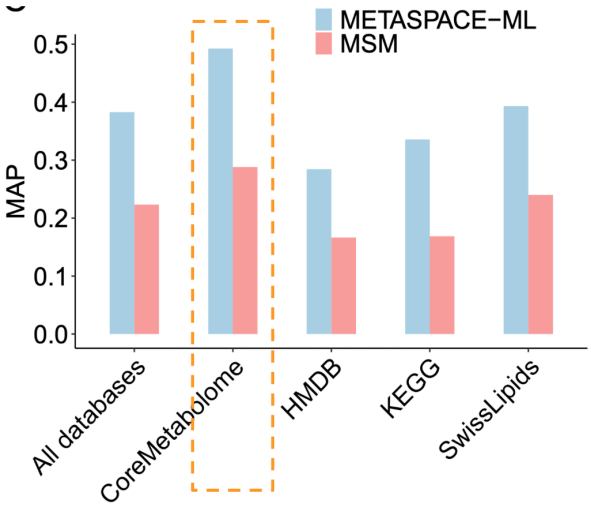


Figure 54: The MAP values across all datasets with the value for the recommended database (CoreMetabolome) highlighted, increasing from 0.3 to 0.5.

Figures 53, 54, 55 show that the ML-version leads to substantial scientific improvements in all three metrics (PR, AUC, MAP, and number of Identified metabolites). Specifically, we have achieved the increase of PR-AUC from 0.25 to 0.45 on average, of MAP from 0.3 to 0.5; the number of metabolites identified at FDR 10% increased by 10-100 new molecules.

KPI-3: Demonstrated resource auto-scaling for batch and stream data processing, validated thanks to data-driven orchestration of massive workflows.

Lithops supplies data-driven partitioning of parallel workloads. Integrating it into the ML-pipeline provides a scalable building block for inputs of variable data size. To evaluate the scalability of the

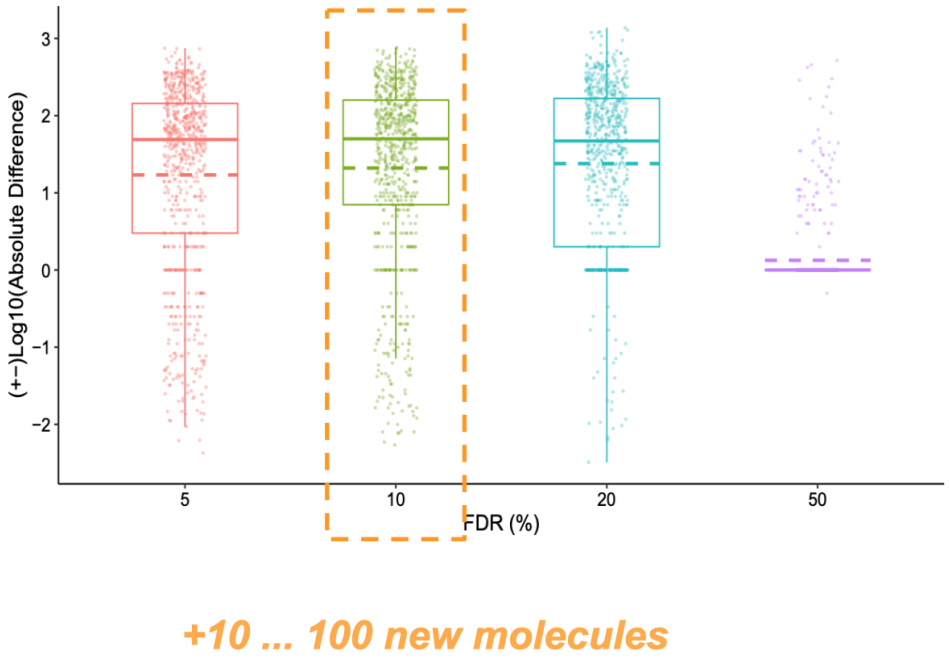


Figure 55: The difference in the numbers of annotations for different FDR levels showing that at FDR 10% (the default value), there are 10-100 more metabolites identified by the ML-version.

ML implementation, we evaluate the two most significant metrics for the use case, **runtime** and **cost**, on increasing input dataset sizes (Table 5).

We compare the runtime of FDR calculation stage, at different inputs. We used the standard METASPACE configuration for Lithops. Table 6 gives the relative speedup, for each input, of the new ML-based pipeline against the production version. The runtime values ranging from 1.7 to 4.58 seconds and only minorly depending on the dataset size validate the scalablity of the system. Table 7 gives the relative speedup, for each input, of the production version of the pipeline against the new ML-based version. Although the new implementation is slower than the old one (1.5-2 times slower), such minor increase in the runtime is actually due to delivering larger numbers of identified metabolites that adds an additional overhead.

Table 6: Runtime for the ML-based metabolite identification, calculated on the production version of METASPACE, in seconds.

Dataset ID	PamDB	CoreMetabolome	LipidMaps	HMDB
ds1	1.80s	3.24s	2.34s	2.51s
ds2	1.69s	2.05s	2.35s	3.10s
ds3	1.29s	1.72s	2.10s	3.26s
ds4	1.74s	1.99s	3.02s	4.58s
ds5	1.72s	2.31s	3.67s	4.25s

On the other hand, we calculated the cost of executing the ML-based identification on our production version of METASPACE and compared it to our old rule-based version. Table 8 shows the absolute values, in cents. Table 9 shows the ratio as compared to the rule-based version. Cost is

Table 7: The ratio of the runtime for the ML-based metabolite identification compared to the old rule-based identification, calculated on the production version of METASPACE.

Dataset ID	PamDB	CoreMetabolome	LipidMaps	HMDB
ds1	2.31	3.7	1.83	1.5
ds2	2.14	1.97	1.59	1.49
ds3	1.94	1.97	1.57	1.43
ds4	1.92	1.96	1.84	1.54
ds5	2.04	1.84	1.53	1.38

pretty much identical with most combinations of the datasets and databases costing 101-110%. Interestingly, this was achieved even in cases where the calculation was slower (e.g. ds5 for HMDB required 1.38 more runtime, yet the cost is 103%).

Table 8: The cost of performing the ML-based metabolite identification, calculated on the production version of METASPACE, in cents.

Dataset ID	PamDB	CoreMetabolome	LipidMaps	HMDB
ds1	1.36ct	2.62ct	5.69ct	6.22ct
ds2	1.48ct	2.50ct	4.74ct	9.23ct
ds3	3.94ct	5.91ct	10.60ct	15.23ct
ds4	10.34ct	19.07ct	46.88ct	86.85ct
ds5	17.24ct	30.68ct	88.76ct	120.16ct

Table 9: The ratio of the cost of performing the ML-based metabolite identification compared to the old rule-based identification, calculated on the production version of METASPACE.

Dataset ID	PamDB	CoreMetabolome	LipidMaps	HMDB
ds1	1.23	1.17	1.1	1.0
ds2	1.19	1.15	0.99	1.04
ds3	1.01	1.06	1.06	1.12
ds4	1.01	1.01	1.02	1.02
ds5	0.99	1.06	1.03	1.03

KPI-5: Demonstrated simplicity and productivity of the software platform validated with real user communities in International Health data Spaces.
We have submitted a manuscript [21] which is currently in review in *Nature Communications*. The ML-based metabolite identification is already available to users on the production version of METAS-

[21] https://www.biorxiv.org/content/10.1101/2023.05.29.542736v2

PACE and is already used by the METASPACE users (Figure 56).

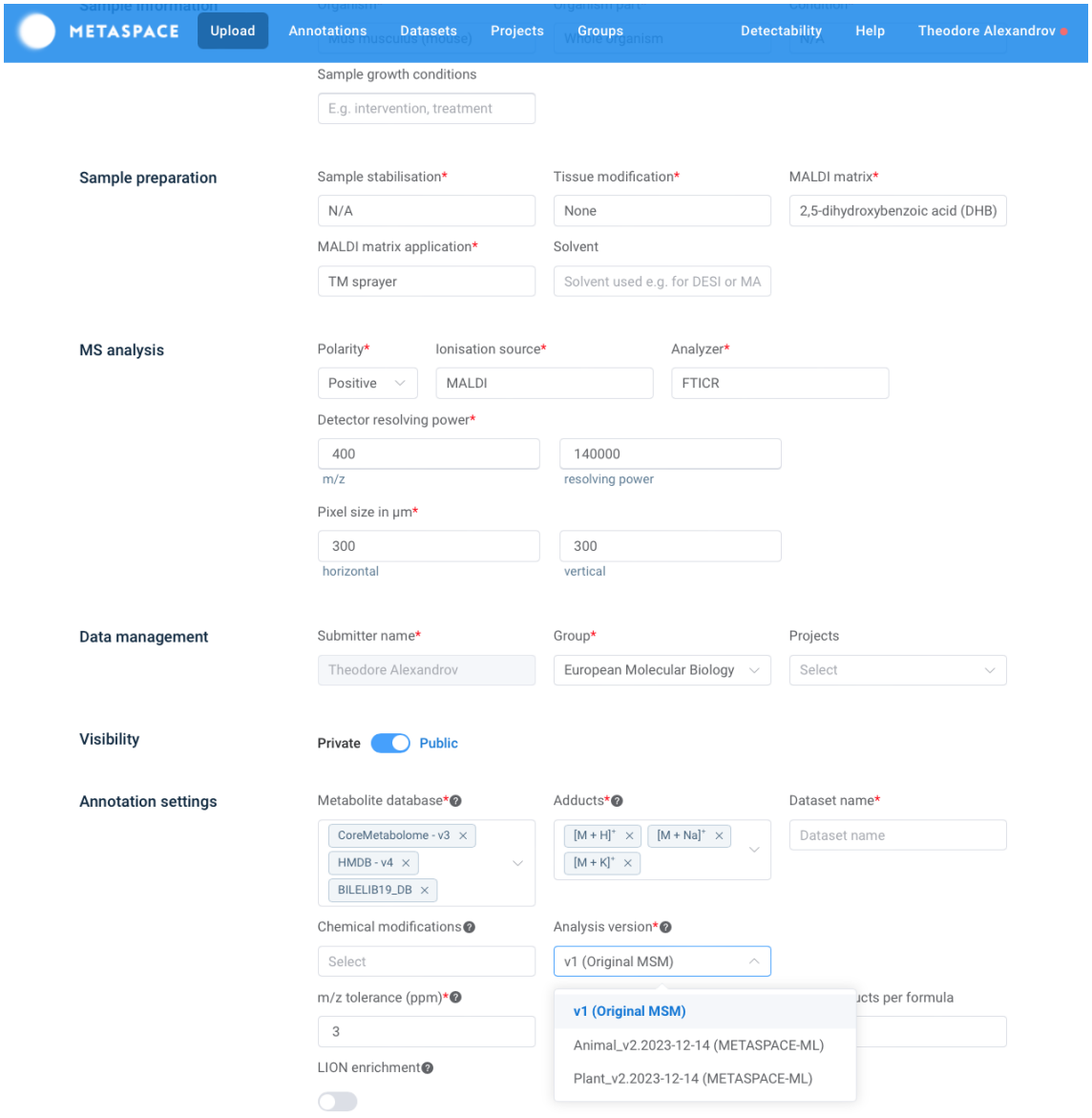


Figure 56: Demonstration for KPI-5, a screenshot from the upload page of the METASPACE (`https://metaspace2020.eu/upload`) demonstrating (bottom part) the ML-based version of metabolite identification developed in Experiment 1 available on production.

7.5.2 Experiment 2 - Confidential computing for safe offloading of ML-based annotation of "dark matter" for private datasets.

Once experiment 1 is finished, experiment 2 will port the existing ML-base pipeline architecture into a confidential trusted environment. Privacy and security will be achieved using SCONE containers over a confidential K8S backend and a Lithops orchestrator. Most OMICs data have strict confidentiality requirements that are not compatible with public cloud services. Running the pipeline at the edge, using on-premise cluster resources, will fit the needs of private workloads.

Experiment 2 is still at its early steps, however we achieved an initial prototype of the integration of Lithops, METASPACE and SCONE containers. We used the KIO testbed for this purpose, whose native K8S deployment served us to implement a potential architecture for the definite version of the use-case. Our final proposal, will integrate the confidential version of the pipeline to process the

private datasets from METASPACE and ingest the results into the METASPACE database and the ElasticSearch engine.

To avoid repeating the previous KPI-s, which are transversal to the whole use-case, we will only include the specification of KPI-4. KPI-4 is strictly correlated with experiment 2 and its future validation steps.

KPI-4: High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.

Although SCONE was integrated into Lithops successfully, the current version of the architecture could not be validated in adversarial contexts. Our next steps include (1) further security stress-tests on the ML-based annotation of "dark-matter" for private sets (2) performance comparisons of the private version of the pipeline against the cloud-based, production one.

8 Conclusions

This deliverable has shown the work done on the NEARDATA project related to the use cases up to M14 of the project. First, we have assessed why each of the use cases are extreme-data cases that require moving the computations near to data. This is summarized in table 10.

Following this initial description of the use cases, we have explained the improvements made so far in the use cases. To achieve such improvements we have implemented a set of data connectors intended to move computations near to where the data is. In some cases, the connectors consisted of integrating NEARDATA architectural frameworks (Lithops, Pravega, and SCONE). In some other cases have been connectors that allow computational pieces to directly access the data, removing middle layers that can be typically found in software stacks such as Apache Spark. Table 11 lists the data connectors developed so far as well as their description and the use cases which are utilizing them.

We have evaluated how those data connectors developed so far perform utilizing the main KPIs described in the Description of the Action (DoA). Table 12 highlights the main KPIs achieved by each of the use cases and quantifies them. For simplicity in the table, we have shortened the KPIs, whose full names are as follows:

- KPI-1: Significant performance improvements (data throughput, data transfer reduction) in Extract-Transform (ETL) phases validated with near-data connectors over extreme data volumes (genomics, metabolomics).
- KPI-2: Significant data 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.

Finally, to make publicly available the work done so far, a Git Hub repository for the NEARDATA project has been enabled. This repository contains the several use cases implemented so far. Table 13 shows the GitHub URL where available. Some of the use cases are not yet ready to be publicly released (as this deliverable is a public release) as they are pending for conference and/or journal publications.

Use-case	Why extreme-data?
Variant-Interactions	The discovery of variant groups associated with the disease involves the analysis of over 15 million variants. Studying all the possible combinations of pairs of variants represents more than 6x1013 tests. Using the Mare Nostrum 4 (MN4) architecture (165,888 CPU - 1.880 GB/core and 208 GPU - 16GB/each), this process is estimated to take approximately 9 days. Ideally, however, we would like to explore combinations with sets of all 23 chromosomes, which would take months.
Surgery	In surgical scenarios, the concept of data-driven computer-guided surgery (CGS) emerges as a solution to aid surgeons during procedures. However, CGS systems encounter various challenges due to the large volume of data they handle. A robust CGS framework must effectively manage large volumes of real-time video data, high-resolution data streams, and temporal information crucial for meaningful guidance. On top of this, we must deal with stringent regulatory compliance and confidential requirements.
Transcriptomics	To build the Transcriptomics Atlas for a given cell or tissue type, the prerequisite is to obtain a set of identifiers of the input files (usually hundreds of files are needed per cell/tissue type). The input data for pipelines are SRA files which contain RNA sequencing data. The total size of the possible subset of available SRA files that need to be processed for the atlas consists of e.g. 20 human tissues is around 10.8 TiB for 8300 files. Moreover, this file must be uncompressed and converted into FASTQ format, increasing the input dataset to 79.8TB of data. Increasing the number of tissues moreover scales the dataset size to hundreds of terabytes.
Genomics	Genome sequence analysis is a compute- and data-intensive task. Current sequencing techniques generate an overwhelming volume of data, with repositories accumulating several petabytes each year [38]. Although HPC installations are commonplace in biology departments, they cannot handle peak demands, reanalyze extensive public datasets generated by large consortia, or minimize the run time needed for specific analyses.
Metabolomics	METASPACE, which is at the center of our work, is strongly positioned as an International Data Space within the field of spatial metabolomics and represents a use-case for extreme data in this field. The large size of spatial metabolomics datasets (1-100 GB), multiplied with the increasing numbers of datasets submitted to METASPACE (10-100 each day) creates a continuously growing amount of data in METASPACE that reached 45 TB of data as of April 2024. Processing incoming datasets as well as analysis of the historic data (e.g. using machine learning-based identification as presented in our Experiment 1) requires performant, highly-scalable, cost-efficient and cloud-focused methods that can be deployed where the data is stored.

Table 10: Summary of why use cases are extreme data

Data connector	Description	Use-cases
HPC connector	Allows to accelerate computations and is part of the XtremeHub	Variant-Interactions (MDR)
Data selector	Selects the data to be used and analyzed	Variant-Interactions (MDR and GWD), Metabolomics
Data preparator	Loads the data in the correct format and gets the relevant information	Variant-Interactions (MDR)
Test of association	Associates the variants with clinic and control	Variant-Interactions (MDR)
Data functional interpretation	Interprets the final execution data	Variant-Interactions (MDR)
Data partitioner	Partitions the data into smaller sets	Variant-Interactions (GWD), Genomics, Metabolomics
Data merger	Merges data coming from the partitions	Variant-Interactions (GWD), Genomics, Metabolomics
Data predictor	Predicts which variants are relevant	Variant-Interactions (GWD)
Federated learning	Integrates SCONE capabilities to enable a secure federate learning environment across hospitals	Surgery, Transcriptomics
Data loader	Allows to access object storage and load intermediate data for further processing	Genomics, Metabolomics
Data shuffling	This connector shuffles the data in the first part of the Genomics use-case using the FASTQ format	Genomics
Near-Data data shuffling	This connector intends to improve the current shuffling connector reducing the volume of data transfers	Genomics
Pravega clients	Event-based clients APIs to connect applications to Pravega	Surgery, Genomics
Pravega GStreamer	Efficiently manage GStreamer media streams in Pravega	Surgery
Data plug connector	Integrates the data plug library to efficiently handle partitions across nodes	Transcriptomics
Lithops connector	Allows to integrate with the Lithops framework	Transcriptomics, Variant-Interactions (GWD), Metabolomics
Data submitter	Allows a user to submit a spatial metabolomics dataset for Lithops-based processing in METASPACE	Metabolomics
Feature computer	Accesses the dataset in the object storage and calculates features necessary for the machine learning model, using Lithops	Metabolomics
FDR estimator	Calculates the False Discovery Rate using our previously published approach but for new features used in the machine learning model, and using Lithops	Metabolomics
Results downloader	Allows a user to access results calculated obtained using the ML-model deployed on production (developed and implemented in Experiment 1)	Metabolomics

Table 11: Data connectors summary table

Use-case	KPI	Results
Variant-Interactions (MDR)	KPI-1	MPI version shows a speed-up of 5x compared to the Apache Spark version which translates into 5x more data ingestion capabilities (we can compare 5x more variants in the same amount of time).
Variant-Interactions (GWD)	KPI-1	We explored the adoption of a GPU component in our HPC data connector to further improve performance on heavy computational tasks. We have seen a potential speed-up of 2.1x times.
	KPI-3	Leveraging Lithops we can now increase the data ingestion capabilities, thus, improving our data throughput. Data throughput has bieen improved as wel as performance. Initial experiments in local environments show a speed-up of 2.83x.
Surgery	KPI-2	With Pravega integration end-to-end IO latency represents only a fraction of the inference latency in the tested models (i.e., 45% lower).
	KPI-3	Combining federated with Scone will auto-scale the training of multiple clients with extreme-date achieving high performance. The orchestrator component auto-scales streaming analytics automatically leveraging Pravega's stream parallelism
	KPI-4	Utilizing SCONE training data and code has been encrypted by making all training inside TEE enclaves. We guaranteed training computations ran correct code not modified by malicious clients. Preliminary results show such confidentiality is ensured.
	KPI-5	NCT estimates that using containerized AI inference applications jointly with Pravega/GStreamer may reduce the combined deployment and data management time for video analytics in $\approx$ 50%.
Transcriptomics	KPI-1	Through the implementation of an early stopper for STAR aligner we can predict whether we should stop processing the alignment with just 10% of the total number of reads.
	KPI-1	We have generated new STAR index on the newest genome release that resulted in (on average) 12-times faster pipeline that has smaller resource (RAM) requirements.
	KPI-1	The FL workflow yields a significant data transfer reduction compared to centralized approaches.
	KPI-1	The FL workflow yields improved data ingestion rates compared to centralized approaches due to inherent parallelism.
	KPI-3	We have determined the best memory per core trade-off in the AWS cloud for the use-case allowing us to choose the most cost-efficient machines.
	KPI-3	Index distribution solution has proved itself viable and scalable for the Transcriptomics Atlas architecture.
	KPI-3	Using spot instances we are able to reduce compute costs by about 50%.
	KPI-4	Data transfer reduction due to FL nature guarantees a high level of privacy. Further integration with SCONE will only improve these guarantees.
	KPI-5	Increased regulatory compliance and data security due to FL nature increases the productivity of the platform.
Genomics	KPI-1	Reduced data partitioning, data transfer (by 200%), and data duplication through the integration of the Dataplug connector, as we avoid re-uploading partitions to the object storage.
	KPI-3	Moving the variant calling pipeline from HPC to serverless with Lithops demonstrates a further optimization of the resources needed for the massively parallelized pipeline thanks to the elasticity and flexibility of cloud services. Thanks to this, the serverless version is x37.46 times faster than the HPC version. In addition, Glider integration on the serverless version reduces execution time by 36% with the full data. The Near-data shuffling connector demonstrates a reduction of the data transfer ensuring the workflow orchestration from the distribution of the actions.
Metabolomics	KPI-1	We have implemented the machine learning-based version of the metabolite identification using Lithops and deployed it on production of METASPACE, thus eliminating the need for data transfer as ML inference can be performed directly in METASPACE.
	KPI-3	The implemented ML-version of metabolite identification allows for resource auto-scaling for datasets of the size ranging from under 1 GB to 20 GB (a common range of datasets in METASPACE). This has been demonstrated in the pilot study as well as on production.
	KPI-4	We have integrated SCONE into Lithops and tested its early usage on a on-premise K8s cluster.
	KPI-5	The ML-based metabolite identification is already available to users on the production version of METASPACE and is already used by the METASPACE users.

Table 12: Highlights of main KPIs achieved by use-cases

Components	Description	GitHub URL
Variant-Interactions use-case	MDR use-case source-code integrated with HPC Data Connector	https://github.com/neardata-eu/MPI-genomics-MDR
	MDR use-case with Apache Spark	https://github.com/neardata-eu/spark-mdr
Lithops backend	Lithops backend for singularity	https://github.com/neardata-eu/lithops-hpc-singularity-and-k8s-backend
Video-streaming benchmarks	Video-streaming benchmarks with Gstreamer and Pravega connectors	https://github.com/neardata-eu/video-streaming-benchmarks
Dataplug	Dataplug python framework	https://github.com/neardata-eu/dataplug
Genomics use-case	Variant calling source code	https://github.com/neardata-eu/serverless-genomics-variant-calling
	Glider	https://github.com/neardata-eu/glider-store
Surgery use-case	Federated Learning Source Code	https://github.com/neardata-eu/nct_tud_fl_demo
	Surgical Pravega GStreamer Demo	https://github.com/neardata-eu/nct_dell_demo_pravega
Transcriptomic Atlas use-case	Federated Learning for Human Genome Variation Analysis	https://github.com/SanoScience/neardata-fl-for-transciptomics
	Transcriptomic Atlas Pipeline	https://github.com/neardata-eu/transcriptomics-atlas-sano
Metabolomics use-case	METASPACE codebase including the implementation of ML-based metabolite identification (Experiment 1)	https://github.com/metaspace2020/metaspace
	ML models from Experiment 1	https://github.com/metaspace2020/metaspace/tree/master/metaspace/scoring-models

Table 13: Use-cases source codes released.

References

[1] BSC-CNS, "Marenostrum 4 supercomputer technical information." https://www.bsc.es/marenostrum/marenostrum/technical-information, 2017.

[2] "Red española de supercomputación (res)." https://www.res.es/es/acceso-a-la-res/, 2024.

[3] "Partnership for advanced computing in europe (prace)." https://prace-ri.eu/hpc-access/calls-for-proposals/, 2024.

[4] BSC-CNS, "Greasy software user's guide." https://www.bsc.es/user-support/greasy.php, 2023.

[5] 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," Scientific Data, vol. 10, no. 1, pp. 1–8, 2023.

[6] 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.

[7] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. De Mathelin, and N. Padoy, "Endonet: a deep architecture for recognition tasks on laparoscopic videos," IEEE transactions on medical imaging, vol. 36, no. 1, pp. 86–97, 2016.

[8] "Greasy software user's guide." https://github.com/rancher/local-path-provisioner, 2023.

[9] "NCBI NIH website." https://www.ncbi.nlm.nih.gov/, 2024.

[10] K. Karczewski and L. Francioli, "The genome aggregation database (gnomad)," MacArthur Lab, pp. 1–10, 2017.

[11] T. gnomAD Consortium, "gnomad v4.0." https://gnomad.broadinstitute.org/news/2023-11-gnomad-v4-0/, 2023.

[12] . G. P. Consortium et al., "A global reference for human genetic variation," Nature, vol. 526, no. 7571, p. 68, 2015.

[13] K. J. Karczewski, B. Weisburd, B. Thomas, M. Solomonson, D. M. Ruderfer, D. Kavanagh, T. Hamamsy, M. Lek, K. E. Samocha, B. B. Cummings, et al., "The exac browser: displaying reference data information from over 60 000 exomes," Nucleic acids research, vol. 45, no. D1, pp. D840–D845, 2017.

[14] N. C. I. (NCI), "The cancer genome atlas (tcga) history," 2017.

[15] M. Goddard, "The eu general data protection regulation (gdpr): European regulation that has a global impact," International Journal of Market Research, vol. 59, no. 6, pp. 703–705, 2017.

[16] L. O. Gostin, "National health information privacy: regulations under the health insurance portability and accountability act," Jama, vol. 285, no. 23, pp. 3015–3021, 2001.

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

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

[19] "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.

[20] "The European Nucleotide Archive in 2022." https://doi.org/10.1093/nar/gkac1051, 2024.

[21] "Statistics regarding data growth in ENA." https://www.ebi.ac.uk/ena/browser/about/statistics, 2024.

[22] A. Dimitromanolakis, J. Xu, A. Krol, and L. Briollais, "sim1000g: a user-friendly genetic variant simulator in r for unrelated individuals and family-based designs," BMC bioinformatics, vol. 20, no. 1, pp. 1–9, 2019.

[23] R. C. Team, "R language definition," Vienna, Austria: R foundation for statistical computing, vol. 3, no. 1, 2000.

[24] "Salmon repository," 2023.

[25] "STAR repository," 2023.

[26] 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.

[27] "DESeq2," 2023.

[28] "SRA-Toolkit repository," 2023.

[29] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council."

[30] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Artificial intelligence and statistics, pp. 1273–1282, PMLR, 2017.

[31] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, et al., "Flower: A friendly federated learning research framework," arXiv preprint arXiv:2007.14390, 2020.

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[33] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," IEEE signal processing magazine, vol. 29, no. 6, pp. 141–142, 2012.

[34] A. Krizhevsky, G. Hinton, et al., "Learning multiple layers of features from tiny images," 2009.

[35] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell, et al., "{SCONE}: Secure linux containers with intel {SGX}," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 689–703, 2016.

[36] 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.

[37] "STAR aligner manual." https://github.com/alexdobin/STAR/blob/master/doc/STARmanual.pdf, 2024.

[38] E. N. Archive, "Statistics." https://www.ebi.ac.uk/ena/browser/about/statistics, 2022.

[39] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: distributed computing for the 99%," in Proceedings of the 2017 Symposium on Cloud Computing, SoCC '17, (New York, NY, USA), p. 445–451, Association for Computing Machinery, 2017.

[40] Q. Pu, S. Venkataraman, and I. Stoica, "Shuffling, fast and slow: Scalable analytics on serverless infrastructure," in 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), (Boston, MA), pp. 193–206, USENIX Association, Feb. 2019.

[41] S. Marco-Sola, M. Sammeth, R. Guigó, and P. Ribeca, "The gem mapper: fast, accurate and versatile alignment by filtration," Nature methods, vol. 9, no. 12, pp. 1185–1188, 2012.

[42] L. Ferretti, C. Tennakoon, A. Silesian, G. Freimanis, and P. Ribeca, "Sinple: Fast and sensitive variant calling for deep sequencing data," Genes, 2019.

[43] A. W. Services, "Amazon s3 object lambda." `https://aws.amazon.com/es/s3/features/object-lambda/`, 2021.

[44] A. W. Services, "Introducing aws lambda response streaming." `https://aws.amazon.com/es/blogs/compute/introducing-aws-lambda-response-streaming/`, 2023.

[45] P. Stuedi, A. Trivedi, J. Pfefferle, A. Klimovic, A. Schuepbach, and B. Metzler, "Unification of temporary storage in the NodeKernel architecture," in 2019 USENIX Annual Technical Conference (USENIX ATC 19), (Renton, WA), pp. 767–782, USENIX Association, July 2019.