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

D2.1 Data infrastructure and data mining framework requirements

Version 1.0

Documentation Information

Contract Number	**101093110**
Project Website	https://extract-project.eu/
Contratual Deadline	30.06.2023
Dissemination Level	Public
Nature	Report
Author	SIX, URV, LRI, BSC, IKL, IBM, MATH
Contributors	SIX, URV, LRI, BSC, IKL, IBM, MATH
Reviewer	IBM
Keywords	Requirement, data, infrastructure, mining

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

Change Log

Version	Description Change
V0.1	Defining ToC and requirements methodology
V0.2	Ready for review version
V0.3	IBM review
V1.0	BSC Final review

Table of Contents

1. Executive Summary

The aim of this "Data infrastructure and data mining framework requirements" deliverable is to provide a list of the requirements for the EXTRACT [1] data infrastructure and data mining framework, and to derive a selection of technologies to be used throughout the project.

Below is the list of the layers and focus areas which were used for the requirements specification that all contribute to the overall data infrastructure and data mining framework specification:

- Data content and metadata layers
- Semantic layer
- Cloud/Edge data staging, and the interconnection with data-mining frameworks
- HPC and AI & big data frameworks
- Data security and privacy

The specification of the collected requirements will be used to drive the work for the other tasks in WP2 "Data Infrastructure and Data Mining Frameworks".

The approach taken for requirements elicitation consists of the definition of the thematic User Stories and then derivation of the corresponding system requirements to support the functional and non-functional levels of the defined User Stories. The User Stories describe on a high-level the desired properties and behaviour of the system for users to be able to fulfil their required tasks around the data infrastructure and data mining framework. The derived requirements directly correspond to providing users with the functionalities they require from the data infrastructure and data mining framework. A ranking of the requirements is also applied followed by the selection of the technologies and tools that implement them.

The document is structured as follows. In section 2, we define the approach that was used for the collection of the requirements. Then, section 3 lists the User Stories identified as part of the user interactions with the data infrastructure and data mining frameworks layer of EXTRACT. Section 4 derives system and software level functional requirements covering all the layers and focus areas of the data infrastructure and data mining framework that were reflected in the User Stories. In section 5 we select technologies and tools to implement the identified requirements. The deliverable is finalised by the conclusions in section 6.

2. Approach to requirements collection

To accomplish the requirement elicitation and provide an adequate number of requirements, the MoSCoW [2] method is used. This method is based on application of prioritization strategies in order to include only the most qualified requirements and avoid an extremely abundant list of requirements. MoSCoW method classifies requirements in four different premises:

- **Must Have**: mandatory requirements that add the main value to the product. These requirements cannot be missed since they compose the product. An example could be the security requirements that ensure the compliance of the product.
- **Should Have**: not mandatory requirements but also add important value to the product. These requirements have a similar impact as must have requirements,

but they are not essential and can be rescheduled. An example could be performance improvement.

- **Could Have**: not mandatory requirements, small impact or none. These requirements represent non-core functionalities that are nice to have but far from essential. An example could be progressive user interface.
- **Will Not Have**: not mandatory, requirements that are left to the backlog. These requirements provide almost no impact for a specific release and will just fall out.

In Figure 1, we can show the overall architecture of the software components in EXTRACT platform, and the relationship among the different layers cited above and the general architecture:

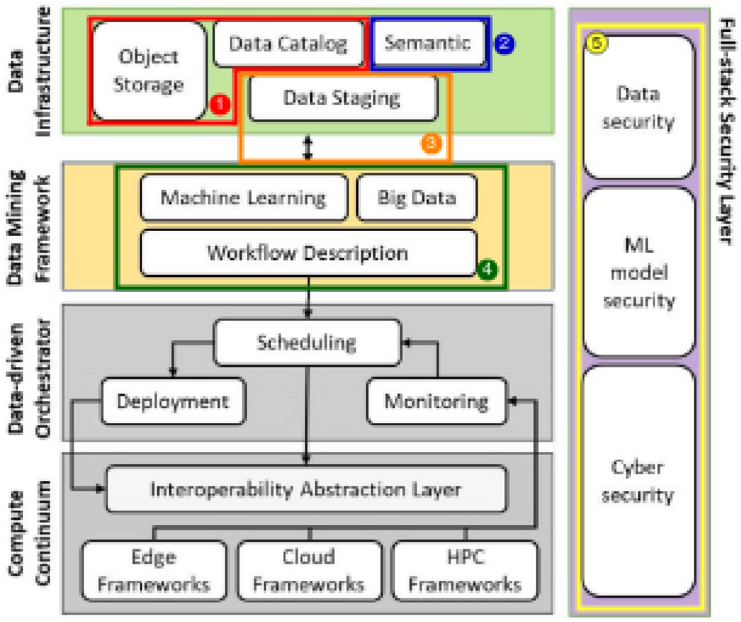

Figure 1: Data infrastructure and data-mining framework layers into software components

The requirements are organised in sections corresponding to the different layers that interact with the data infrastructure of the EXTRACT platform, namely:

1. Data content and metadata layer
2. Semantic layer requirements
3. Cloud/Edge data staging layer and interconnection with data-mining frameworks
4. HPC and AI & big data framework
5. Data security and privacy

Getting deeper into the data infrastructure and data-mining framework layers, figure 2 is useful to show the relationship between components:

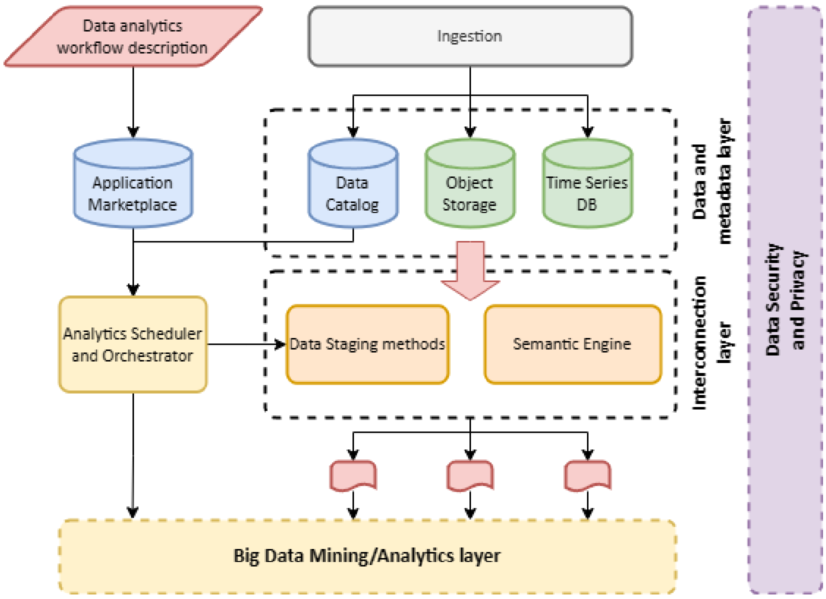

Figure 2: Data infrastructure and data-mining framework layers

Based on the requirements obtained, we follow with a collection of technologies to help their fulfilment, as well as discussions on new developments to adapt them to EXTRACT.

3. User stories

The user stories collected in this document will determine the requirements that the proposed technologies must meet. User stories can be of two types:

- **User-driven**: they specify a functionality to be met, improvement, need, etc. from the point of view of one of the end user personas of the use cases.
- **Technology-driven**: they are functionalities, improvements, proposals, etc. that are to be fulfilled from a technical point of view.

3.1. US_1: PER dynamic emergency plans

As a Venice emergency controller, I would like to have dynamic emergency plans issued to all current occupants of a hazard area within a time limit.

This story is narrowing the PER use-case from D1.1 to the data mining level. The general use-case of PER includes dealing with collection of data from multiple sources, generate and update (if necessary) the personal evacuation plan, and notify the user via their phones. From the data infrastructure point of view the PER use case requires the critical issues are on related on the collection. Curation, access the data mining level, the core user story is the timely generation of the evacuation plans on the base of the condition of the ground and updating them until occupants safely reach the waiting area. These plans are intended to be generated by a machine learning model based on Reinforcement Learning (RL). Given that, it is expected that some big-data processing may be required, before and/or after the inference/training, all subject to the same timing constraints.

3.2. US_2: TASKA ML workflow definition

As a radio astronomer, I would like to define a workflow with ML models that process my observation data under [real time] constraints.

This particular story is a reduction of several TASKA cases (A, C, D, E) to the data mining level. For example, the general TASKA-A case, as described in D1.1 (UnDysPuted system), involves reducing the incoming stream of raw observation data into a set of non-overlapping sequences, each representing a phenomenon at varying timescales. The TASKA-C case builds on TASKA-A and adds additional processing steps at large scale such as rebinning, calibration and image construction. TASKA-D leverages another ML model for identification of faint sources. TASKA-E adds beyond TASKA-C additional large-scale processing of dynamic spectrum extraction (DynSpecMS). At the data mining level, these boil down to defining a time-constraint workflow that includes serving one or more models (e.g., phenomena detection model, faint source detection model) and pre/post big-data computations at a rate that matches the incoming data rate, all subject to the same timing constraints. Unlike the PER use-case, however, these workflows are expected to vary per each astronomer's requirements.

3.3. US_3: Data storage

As an EXTRACT application developer, I will need to store huge amounts of data (extreme data).

Extreme data refers to large volumes of information that need to be managed and stored efficiently. The users' data sets can be extremely large, ranging from terabytes to petabytes and even exabytes of information.

Choosing the right data storage system is critical to ensure that information can be stored, accessed, and processed quickly and efficiently. The objective is to select the technology(s) most appropriate to the needs and integrate it into the EXTRACT platform.

3.4. US_4: Metadata management

As EXTRACT app developer, I can generate, store, query, and share metadata about my data sets at scale.

App developers can generate and store metadata about their data in a meta-data catalogue, that provides schema-free registration of the metadata records, allowing users to express any required structured information about the data. Users can remotely query the meta-data catalogue to discover the meta-data records. Users can share the meta-data records with other users.

Metadata will play an important role in the EXTRACT platform, enriching users' knowledge of the data stored on the platform.

3.5. US_5: PER semantics approach

As an EXTRACT urban application developer, I would like to have a single service of information that represents the entire state of the city and its inhabitants, in a timely and scalable fashion.

The semantic approach to Urban Digital Twin allows for the integration and harmonization of heterogeneous urban data by providing a common vocabulary, data models, and ontologies. This facilitates a holistic view of the city, enabling seamless data exchange, integration, and analysis within the digital twin.

In particular it supports:

- Contextual Understanding of Urban Data
- Interoperability and Collaboration
- Real-Time Monitoring and Simulation
- Urban Knowledge Discovery
- Citizen Engagement and Empowerment

The general use-case of PER includes dealing with collection of heterogeneous data using ETL process to extract data from disparate sources, transform it into a common format (e.g., RDF), and load it into the Knowledge Base.

The semantic approach allows semantic inference on the data according to the ontology adopted. Moreover, it supports a meaningful continuous urban status retrieval that will be exploited by the ML module to generate the personalized evacuation paths at time of hazard and notifying the occupant's phones. Such semantic-aware information retrieval should be timely and continuous until occupants safely reach the waiting area. These paths are intended to be generated by a machine learning model based on Reinforcement Learning (RL). Given that, it is expected that some big-data processing may be required, before and/or after the inference/training, all subject to the same timing constraints.

3.6. US_6: Data staging tool

As an EXTRACT application developer, I would like to be able to define data preparation steps that should be executed automatically and at scale before the data mining operations.

When we talk about "Data Staging", we refer to a process in which data is collected, prepared and transformed for its analysis. This requires a specific technology to carry out this process effectively.
The first step in the data staging process involves loading the data from a storage system. This can be a database, flat file, cloud source, or other data source. The technology choose must be able to efficiently connect to and extract data from these sources.
Once the data has been loaded, certain transformations must be performed to prepare it properly. These transformations can include data cleansing, integration of different data sets, format normalization, data aggregation, and other similar operations. The technology selected should have the capabilities to perform these transformations efficiently and flexibly.
Once the data has been transformed and prepared, the next step is to deliver it to the data mining frameworks. Data-processing workflows may include machine learning algorithms, statistical analysis, natural language processing, or other techniques to discover patterns, make predictions, and extract insights from data. The technology you choose must have the ability to provide an interface or a way to connect to these data mining methods so that the prepared data can be used in the analysis.
In short, the technology to select and improve must have the ability to load data from the storage system, perform necessary transformations to prepare the data, and efficiently deliver it to data mining methods.

3.7. US_7: Dynamic partitioning tool

As an EXTRACT application developer, I would like to have my data staging operations automatically optimized for performance at scale.

Traditionally, if we have several processes working in parallel performing data staging on a very large file, all these processes will have to download the entire file of interest, load said file into their memory, and then statically partition these files and keep the block, discarding the other data. This has two major drawbacks:

- Download in each worker the entire file in its entirety (+ latency, + data travel)
- Load the entire file into the worker's memory (+ memory usage)

We want to optimize the data load in each of the workers, in such a way that each worker only downloads the data that is strictly necessary to process its block (instead of downloading the entire file). This optimization will mean an improvement in processing speed and a decrease in necessary resources and unnecessary workload.

Since EXTRACT use cases needs lo load huge amount of data from storage, dynamic partitioning gains importance and necessity into the project.

3.8. US_8: Data Security

As an EXTRACT application developer or security owner, I would like to protect secret information from being exposed to unauthorized entities (externally or internally) while avoiding information tampering.

During all the life of data within EXTRACT, data collection, staging, processing and analysis, the data protection requirements should be met, especially for the confidential data. Although many solutions are available, in EXTRACT we will deal with extreme data, to this end we need to choose the technologies with the best performance and which give more accurate results. During the project we will conduct experiments to benchmark the available technologies, however we will need the data characteristic whenever available to include in our experiments (e.g., Data Volume and Frequency).

3.9. US_9: Data Privacy

As an EXTRACT application developer or security owner, I would like to allow the appropriate anonymization/de-identification of sensitive information.

Together with data security, data privacy requirements also should be met, and we can distinguish this user story from the previous one (Data Security) by the stockholders, technologies and the data in question. Data Privacy do not include confidential data; however it is concerning private data which exposing it is not the problem but connecting it to a person or an entity is. We want to implement the appropriate anonymization/de-identification techniques for data obfuscation with the aim to hide private identities as well as other Personally identifiable information (PII) or sensitive information, to secure data storage and data sharing and to properly handle the user consent.

3.10. US_10: Model and Computation Protection

As an EXTRACT ML developer / security owner, I would like to have my models protected w.r.t. data privacy and security.

Models are not completely defensible for privacy and security. Therefore, we will address the models' security also according to the use (internal or external to the EXTRACT environment) of data for ML. In addition, the design of the security solutions needed to improve the security of the Machine Learning models and applications against the adversarial threats of Evasion, Poisoning, Extraction, and Inference will be addressed.

4. Requirements

4.1. Data content and metadata layers

ID	Requirement name	Requirement description	Priority	User Story ID
CM_1	High availability	We need the data storage system to offer high availability of the same, being able to access them at the moment in which it is required.	MUST	US_3
CM_2	Scalability	The storage system is required to be scalable, since handling extreme data will require having to record huge amounts of data.	MUST	US_3
CM_3	Durability	It must be guaranteed that there will be no data loss, that is, that the stored data will remain forever in the system unless the user decides to explicitly delete it.	MUST	US_3
CM_4	Storage resource efficiency	Efficient use of available storage resources.	SHOULD	US_3
CM_5	Integration with applications and services	It is necessary that the chosen storage system has integration (compatible API) with the frameworks that are going to be used in EXTRACT.	MUST	US_3
CM_6	Interoperability across the continuum	The data storage system must be interoperable throughout the entire continuum (Cloud, HPC and Edge)	MUST	US_3
CM_7	Record metadata in the storage system itself	Have the ability to record metadata in the storage system itself.	COULD	US_3
CM_8	Regulatory compliance and security	It must be ensured that the storage system complies with regulations and specific safety	MUST	US_3

		standards. The confidentiality and integrity of the data must be guaranteed.		
CM_9	Meta-data catalogue	EXTRACT MUST provide meta-data catalogue with user-defined schema.	MUST	US_4
CM_10	Data discovery	EXTRACT MUST allow data discovery of the data via meta-data catalogue based on the characteristics of the user data.	MUST	US_4
CM_11	Meta-data sharing	EXTRACT SHOULD allow meta-data to be shared among users.	SHOULD	US_4
CM_12	Authn/z access to meta-data	EXTRACT MUST provide authentication and authorization mechanism for accessing meta-data.	MUST	US_4
CM_13	RESTful API for meta-data access	EXTRACT MUST provide meta-data handling service via RESTful API.	MUST	US_4
CM_14	Programmatic access to meta-data catalogue	EXTRACT SHOULD provide user application developers with libraries for accessing meta-data catalogue via its API.	SHOULD	US_4
CM_15	High availability of meta-data catalogue	EXTRACT meta-data catalogue MUST provide highly available service.	MUST	US_4
CM_16	Scalability of meta-data catalogue	EXTRACT meta-data catalogue MUST be scalable, since handling extreme data requires having to catalogue large amounts of data.	MUST	US_4
CM_17	Durability of meta-data catalogue	EXTRACT meta-data catalogue MUST guarantee that there will be no meta-data loss, that is, that the stored meta-data will remain forever in the system unless the user decides to explicitly delete it.	MUST	US_4
CM_18	Secure meta-data catalogue	EXTRACT meta-data catalogue MUST provide users secure access to the meta-data entries store in it.	MUST	US_4
CM_19	Temporal-based information retrieval	EXTRACT MUST support data/information retrieval on temporal logic basis	MUST	US_3

4.2. Semantic layer

ID	Requirement name	Requirement description	Priority	User Story ID
SL_1	Integration of Data across Disparate Data Sources	It must integrate data across different data sources using ontology as a common data model and SQL and/or SPARQL [3] enroute to extract richer Information Representation of the city.	MUST	US_5
SL_2	Transform (static) heterogeneous data sources into a common format and logic	It must transform information coming from different sources with ETL process in an RDF representation according to the ontology selected.	MUST	US_5
SL_3	Semantic Search	Implement a semantic search engine enhanced with semantic indexing and ranking techniques, to enable efficient and context-aware information retrieval from the knowledge base. The search engine is not intended to work for Real Time interaction.	COULD	US_5
SL_4	Querying and Retrieval	The layer must efficiently retrieve the status of the city using SPARQL/SQL anytime each agent requests it regardless of how many they are. It is intended to work on demand and require high availability, very low response time (close to real time).	MUST	US_5
SL_5	Supports efficient ontology	The layer has to support the implementation of families of Description Logics (DLs) specifically designed to keep all reasoning tasks polynomially tractable in the size of the data and is thus suitable for OBDA.	MUST	US_5
SL_6	Semantic Reasoning	Apply reasoning engines, to perform inference and logical deductions based on the axioms	MUST	US_5

		and rules defined in the ontology. Reasoning enhances the semantic capabilities of the knowledge base, enabling automated inference and intelligent querying.		

4.3. Cloud/Edge data staging and integration with data mining frameworks

ID	Requirement name	Requirement description	Priority	User Story ID
DS_1	Access to storage system (extraction)	The technology used must provide the necessary directives to access the data source and be able to load the corresponding data into its working memory.	MUST	US_6
DS_2	Transformation of the obtained data	Once the data has been extracted, a series of transformations can be applied to clean, filter, combine, or aggregate data as needed.	MUST	US_6
DS_3	Loading the data towards data-mining framework	Once the data is transformed, it must be able to deliver data already processed to the corresponding data mining methods.	MUST	US_6
DS_4	Dynamic scaling of resources based on source size	The technology used should have the ability to scale the necessary resources depending on the size of the data, so that in an elastic way they choose the necessary computational resources to be able to face these workloads. Also needs to work across the continuum, wherever is most efficient.	SHOULD	US_6
DS_5	On-the-fly partitioning	EXTRACT MUST allow data ingestion to perform on-line partitioning, avoiding expensive preprocessing and duplicating stored data	MUST	US_7
DS_6	Data formats description	EXTRACT will need to offer a way to specify the structure of the data, which will allow access to the data using dynamic partitioning.	MUST	US_7

4.4. HPC and Big Data & AI frameworks

ID	Requirement name	Requirement description	Priority	User Story ID
DM_1	Big Data computation support	Provides facilities for big data computation such as Map-Reduce.	MUST	US_1, US_2
DM_2	Machine Learning model training support	Provides support for training ML models, at least for popular formats (PyTorch / ONNX / ...).	MUST	US_1, US_2
DM_3	Machine Learning model serving support	Provides support for serving ML models (beyond direct inference) to allow distributed deployment and scalability.	MUST	US_2
DM_4	Integration with Data Staging	Allows seamless integration with data staging, in particular the catalogue and meta-data generation for all created/computed datasets.	MUST	US_1, US_2
DM_5	Workflow and dataflow constraints	Allows specification of constraints over workflow (e.g., to allow distribution) and dataflow (to allow timely processing of data and data security/privacy enforcement).	SHOULD	US_1, US_2

4.5. Data security and privacy

ID	Requirement name	Requirement description	Priority	User Story ID
Internal requirements				
SP_01	Data volume	Provides information about the data volume to be secured, in order to choose the most suitable algorithm.	SHOULD	US_8, US_9
SP_02	Data velocity	Provides information about the expected frequency/rate of the data need to be secured, since the data rate is crucial for the security technique to be chosen.	SHOULD	US_8, US_9
SP_03	Data variety	Provides information about the expected type of the data to be processed so as to find the most	SHOULD	US_8, US_9

		appropriate way to deal with them.		
SP_04	Model structure	The model structure will influence the security solution chosen for instance a machine learning based solution (e.g., Multiparty Computation).	MUST	US_8, US_10
SP_05	Processing type	The type of processing is key for identifying the technology to be used, in particular if data should be locally o distributed.	MUST	US_10
SP_06	Other processing	This requirement will provide information on other processing (e.g., Data hashing, cleaning, …) affecting the data to be secured.	SHOULD	US_8, US_10
External requirements				
SP_07	Data confidentiality	When treating data, the platform needs to know which data are confidential and in which level.	MUST	US_8
SP_08	Data sensitivity	When treating data, the platform needs to know which data are sensitive so as to effectively deal with them also in term of data anonymization.	MUST	US_9
SP_09	Model security	When applying the ML model, the platform needs secure the model parameters and output.	MUST	US_10
SP_10	Performance	EXTRACT should describe the expected level of performance time and if it is online or offline for each data/model combination.	SHOULD	US_8, US_9, US_10

5. Technologies proposal

5.1. Data content and metadata layer

5.1.1. Object Storage

Object Storage is the most appropriate solution to the requirements generated in US_3. Object storage is a form of data storage in which files and associated metadata are stored as individual objects. Unlike traditional block or file-based storage, where data is organized in hierarchical folder and file structures, object storage organizes data into individual objects that are identified by a unique key.

Below we can see how Object Storage meets all the requirements:

Requirement ID(s)	Justification
CM_1	High availability: Object storage is designed in such a way that it allows data replication, supports fault tolerance even in case of hardware or network failures and allows the option to distribute the data geographically.
CM_2	Scalability: due to its ability to add additional nodes as needed to increase system capacity and performance.
CM_3	Object Storage offers durability due to its data replication and use of failover techniques, ensuring data integrity and availability over time.
CM_4	Object Storage achieves resource utilization efficiency through intelligent storage, dynamic scalability, fast access, simplified management, and lower impact of failures.
CM_5	Object Storage typically uses a standard API, such as Amazon S3 or OpenStack Swift, that allows applications to interact with object storage in a consistent manner. This API is widely supported and used by a variety of cloud service providers and storage solutions.
CM_6	Object Storage can be deployed on the continuum, as it supports both edge (MinIO, OpenIO, EdgeFS), cloud (Amazon S3, Google Cloud Storage, Microsoft Azure Blob Storage) and HPC (Lustre, Ceph) support. This feature will allow us to store data in the best place within the entire continuum.
CM_7	Object Storage offers its metadata system that adds knowledge to the data itself within the platform. Although this is not a MUST requirement of the storage system, it may be desirable for the storage system itself to offer these features.
CM_8	Object Storage is regulatory compliant and provides security through measures such as access control, data encryption, replication, durability, auditing, and regulatory compliance. These guarantee the protection of the stored data and comply with confidentiality, integrity and availability requirements.

In summary, Object Storage offers a robust and reliable solution for efficient data management, which will cover the requirements fitting the EXTRACT use cases. Besides those benefits, it has limitations that need to be considered for its usage: like updates replace the objects, cannot seek, cannot append, etc.

5.1.2. Nuvla

For user story "US_4 Metadata management", Nuvla [4] is an ideal technological solution for providing a metadata service and fulfilling the requirements of the user story. Nuvla offers a comprehensive and scalable platform that enables users to generate, store, query, and share metadata about their datasets effectively.

One of the key advantages of Nuvla is its ability to handle metadata at scale. With Nuvla, users can manage and organize metadata for a vast number of datasets without any limitations on the size or complexity of the data. Whether it's a small collection or a massive dataset, Nuvla's robust architecture (based on horizontally scalable stateless API services and backed by highly scalable Elasticsearch search engine) ensures efficient storage and retrieval of metadata records.

Nuvla's metadata catalogue is designed to provide schema-free registration of metadata records, allowing users to express any required structured information about their data. This flexibility empowers users to define and adapt metadata schemas according to their specific needs. By eliminating rigid schema constraints, Nuvla enables users to capture rich and diverse metadata, facilitating a more comprehensive understanding of the underlying datasets.

Furthermore, Nuvla offers powerful remote querying capabilities for the metadata catalogue. Users can perform advanced searches and discover metadata records based on various criteria, such as keywords, tags, or specific data attributes. The ability to remotely query the metadata catalogue ensures that users and third-party services can efficiently explore and identify relevant metadata records, saving time and effort in data exploration and analysis.

Nuvla also excels in facilitating metadata sharing between users. By leveraging its collaborative features, Nuvla enables seamless sharing of metadata records with other users or teams. Users can control the access privileges and define sharing rules, ensuring secure and controlled sharing of metadata. This collaborative approach promotes knowledge sharing, enhances collaboration, and fosters a more data-driven and informed decision-making process.

Requirement ID(s)	Justification
CM_9	Meta-data catalogue: Nuvla provides meta-data catalogue for storing information about user's data: objects, records, sets.
CM_10	Data discovery: Through its query language, Nuvla allows users to perform advanced searches and discover metadata records based on various criteria, such as keywords, tags, or specific data attributes.
CM_11	Meta-data sharing: On Nuvla, users can share meta-data objects, records and sets among themselves thanks to the versatile ACLs mechanism.

CM_12	Authn/z access to meta-data: Nuvla provides strong 2FA authentication and enforcement of ACL to handle the authorization when accessing data objects/records/sets in the meta-data catalogue.
CM_13	RESTful API for meta-data access: Nuvla provides RESTful API for remote access to the meta-data catalogue.
CM_14	Programmatic access to meta-data catalogue: Nuvla provides well defined API and user level libraries for integration of the service with third-party services/applications.
CM_15	High availability of meta-data catalogue: The API service behind the meta-data catalogue is built as the horizontally scalable layer out of stateless independent services. The load balancer is placed as the layer in front to balance the incoming requests.
CM_16	Scalability of meta-data catalogue: Nuvla uses a cluster of Elasticsearch instances for storage of the data records, objects and sets that constitute of the metadata catalogue.
CM_17	Durability of meta-data catalogue: Nuvla uses a cluster of Elasticsearch instances with index replication for storage of the data records, objects and sets that constitute of the metadata catalogue. The Elasticsearch is also configured to perform periodic backups to S3 service.
CM_18	Secure meta-data catalogue: Nuvla provides 2FA with TLS.

In summary, Nuvla's comprehensive features, scalability, schema-free registration, remote querying capabilities, and collaborative nature makes it the most appropriate technological solution for providing metadata services. It empowers users to effectively generate, store, query, and share metadata about their datasets at scale, enabling them to harness the full potential of their data resources and accelerate data-driven insights and innovation.

5.1.3. InfluxDB

InfluxDB [5] is an open-source time series database designed to handle high write and query loads for time-stamped data. It is built to efficiently store, retrieve, and analyze time series data, which typically consists of data points associated with timestamps.

Here's a technical description of InfluxDB:

- **Data Model**: InfluxDB organizes data using a key-value pair approach. It uses the concept of a "measurement" to represent a set of data points that share the same measurement name. Each data point is uniquely identified by a timestamp and associated with a measurement, tags, fields, and optionally, a retention policy.
  - **Measurement**: A measurement is a logical container for related data points. For example, you could have a measurement called "temperature" to store temperature readings.

- **Tags**: Tags are key-value pairs attached to data points, providing metadata for efficient filtering and indexing. They are typically used to identify attributes of the data, such as sensor or location information.
- **Fields**: Fields contain the actual data values associated with a data point. They can be numeric, string, or boolean values.
- **Timestamp**: Each data point has an associated timestamp, indicating when the measurement was made.

- **Time Series Data Storage**: InfluxDB stores time series data in a structure called a "shard." A shard is a self-contained data structure that contains a subset of the time series data. Shards are created based on configurable time intervals, enabling efficient data retrieval and retention policies.
- **High Write and Query Performance**: InfluxDB is optimized for high write and query loads. It achieves this through various techniques, such as a write-ahead log, which ensures durability and efficient disk I/O, and a memory-mapped cache that reduces disk I/O operations.
- **Query Language**: InfluxDB provides its own query language called InfluxQL (Influx Query Language) for data retrieval and analysis. InfluxQL supports various functionalities like aggregation, filtering, downsampling, and joining data from different measurements. Additionally, InfluxDB also supports Flux, a more powerful and flexible query language.
- **Retention Policies:** InfluxDB allows the definition of retention policies to manage the lifetime of data in the database. Retention policies specify the duration for which data is stored in a database, as well as the duration of data replication across InfluxDB nodes.
- **Integrations and Ecosystem**: InfluxDB integrates with a wide range of tools and platforms, making it suitable for different use cases. It has libraries and client APIs available for popular programming languages, and it supports integrations with data visualization tools like Grafana, as well as alerting and monitoring systems.

Overall, InfluxDB provides a robust and scalable solution for managing time series data, with a focus on high performance, efficient storage, and powerful query capabilities.

The decision to adopt InfluxDB is grounded on the fact that is already adopted by the Venice Control Room. This would enhance the possibility of a future exploitation of the project result at the pilot site.

Requirement ID(s)	Justification
CM_1	High availability: InfluxDB allows for a clustered InfluxDB installation which consists of two separate software processes: data nodes and meta nodes.
CM_2	Scalability support
CM_3	Durability: the cloud version of InfluxDB allows replicates of all data in the storage tier across two availability zones in a cloud region, automatically creates backups, and verifies that replicated data is consistent and readable. Moreover The Write Ahead Log

	(WAL) retains InfluxDB data when the storage engine restarts. The WAL ensures data is durable in case of an unexpected failure.
CM_4	Storage resource efficiency: InfluxDB allows writing raw data to one bucket with a small retention policy. Then it is possible to run a task to determine if values have changed and only write those changes to a new bucket with a longer retention policy.
CM_5	Integration with applications and services: InfluxDB API provides a simple way interact with the database. It uses HTTP response codes, HTTP authentication, JWT Tokens, and basic authentication, and responses are returned in JSON that allows a standardised integration with applications and services.
CM_19	Temporal-based information retrieval: InfluxDB is a Time Series Database, a database optimized for time-stamped or time series data. InfluxDB is optimized for measuring change over time and to retrieve information on the base of timestamp efficiently.

5.2. Semantic layer

In order to set up the semantic layer, a mix of technologies has been identified. In particular according to the literature the hybrid architecture that combines NoSQL Time Series Database (TSDB) and Graph seems to be recommended for performance requirements. However, in case the approach results not aligned with the application requirements (e.g. intensive and timely data extraction), other solutions will be explored.

There are a variety of approaches to integrating time series data with data stored in a RDF database, involving query rewriting, property value functions, relying on SQL-based data integration, SQL based data lakehouses such as Dremio and client-side integration of data.

In our context we propose a client-side integration approach that uses SPARQL to query static urban information context, which is used to determine what data to extract from the time series database InfluxDB.

The decision to adopt InfluxDB is grounded on the fact that is already adopted by the Venice Control Room. This would enhance the possibility of a future exploitation of the project result at the pilot site. The Knowledge base will be implemented with Virtuoso, an open source (GPL v2) multi-model hybrid-DRBMS database engine to manage RDF triples and support semantic driven heterogeneous data integration. Virtuoso supports multiple domain ontologies and semantic queries in SPARQL. It includes Fine-grained Attribute-Based Access Control (ABAC) in addition to typical coarse-grained Role-Based Access Control (RBAC) according to SQL-standard. Moreover, the possibility to ingest static heterogeneous data into Virtuoso requires ETL tools. A well-defined approach adopted by Snap4City developed by University of Florence includes the combination of Kettle + Karma. Such solution requires the introduction of a MySQL in the workflow since Karma is able to map a model based on ontology from a MySQL table to RDF. Another option can be focused on using dedicated tools for different kind of datasets such as TripleGeo that is able to translate geographic information (SHP file) into RDF.

The related user story US_5 will be addressed with the implementation of the semantic layer. The requirements with respective explanations appear in the next table:

Requirement ID(s)	Justification
SL_1	Integration of Data across Disparate Data Sources. Virtuoso manages multiple domain ontology thus data coming from different sources can be accommodated according to the related ontological concept. Eventually the data generated by the sensors and EMA can be integrated in VIRTUOSO as well using dedicated class (e.g. Observation).
SL_2	Transform (static) heterogeneous data sources into a common format and logic. This procedure can be implemented once and is used to inset the contextual and state info into Virtuoso. Solutions like TripeGeo or Kattle+Karma will be considered.
SL_3	Semantic Search. Dedicated API will be implemented exploiting SPARQL + TSDB query combination mechanism.
SL_4	Querying and Retrieval. The semantic layer will extract the information implementing a dedicated API that will perform multiple queries from KB and TSDB and provide the results in GeoJSON/JSON to the AI module.
SL_5	Supports efficient ontology. Virtuoso is able to support the implementation of multi-domain ontology to accommodate different kind of data (e.g. geographic data)
SL_6	Ontology Reasoning Virtuoso implement an efficient semantic reasoning to enrich the KB with the missing relations according to the ontology definition

5.3. Cloud/Edge data staging and integration with data mining frameworks

5.3.1. Lithops

Lithops [6] is a Python framework that covers most of the requirements that are a result of the US_6 user story.

Lithops is a multi-cloud serverless computing framework. It allows to run unmodified local Python code at massive scale in the main serverless computing platforms.

Lithops provides great value for data-intensive applications like Big Data analytics and embarrassingly parallel jobs.

Also, Lithops facilitates consuming data from object storage (like AWS S3, GCP Storage or IBM Cloud Object Storage) by providing automatic partitioning and data discovery for common data formats like CSV.

We can see in the next table in what way Lithops covers the requirements associated with US_6.

Requirement ID(s)	Justification
DS_1	Lithops provides the directives to read/write data in Object Storage. Since object storage has been proposed as the data storage technology, Lithops is the perfect tool to be able to consume this data.
DS_2	Lithops parallel workers support Python, so a lot of popular libraries can be used to clean, filter, combine, or aggregate data as needed.
DS_3	Adaptations in Lithops will be performed during the course of EXTRACT to facilitate the delivery of the processed data to the pertinent methods.
DS_4	Since dynamic provision of resources is not a mandatory requirement, Lithops gives facilities to easily change the size of workers and chunks as necessary.

5.3.2. Dynamic partitioning tool

We will develop new technologies to aid data partitioning during ingestion and staging into the EXTRACT platform. We have started devising of a tool that provides dynamic data partitioning to be ingested from Object Storage. The development of this tool will suppose one of the research lines contributing to the EXTRACT project.

The related user story US_7 will be covered with the development and integration of this dynamic partitioning tool. The requirements with respective explanations appear in the next table:

Requirement ID(s)	Justification
DS_5	The tool will be especially designed to support on-the-fly partitioning for extreme data.
DS_6	The tool will enable an extensible model to describe data formats and their partitioning strategies.

As dynamic partitioning is a novel technology proposal, our aim in these next paragraphs is to explain in a understandable way the most relevant issues about the dynamic partitioning.

The dynamic partitioning tool will provide advantages over the classical approach (static partitioning).

By means of an example, we can show the essence of the improvements that dynamic partitioning will bring.

Imagine that we want to count the number of words inside a large text file. Since the file is very long, doing it in parallel would be infeasible, since the processing time would be very long.

As a solution, we propose then to use Lithops to do word counting in full parallel. With Lithops, we could program N remote functions that would run concurrently, and each of the remote functions would download the file (stored in Object Storage), count the words in the region of the text that corresponds to it and return the partial word count in its text block. Finally, we would do a reduction and, adding up the partial word counts, we would get the total word count of the huge text file.

However, we can observe that this flow is inefficient. So far, the N remote functions have to download the whole file from Object Storage, so that once loaded into memory, the region of interest is selected and the rest of the file is discarded. On the other hand, creating partitions of the appropriate size requires a costly process to run before the actual computation that generates new files (duplicating data on storage) only useful for a particular job. Static partitioning implies a pre-processing step that does the partitions sequentially and uploads data (in multiple objects, partitions) again to storage.

The proposal is to allow each of the N remote functions to download from Object Storage only the region of data it needs (HTTP-range requests). This will benefit in several ways:

- The download volume will be lower because each worker will download only the data it needs.
- The Object Storage server will have less workload.
- The memory consumed by each worker will be lower since it will not be necessary to store the whole file in memory.

To allow the dynamic partitioning of a file, it is necessary to specify the file format in such a way that the partitioning policy can be defined (known as cloud-native data formats). The definition of the cloud-native data format will allow to indicate to the workers which methodology they will have to follow to download the corresponding data region.

5.4. HPC and Big Data & AI frameworks

The data mining framework is a layer of EXTRACT that provides the building blocks for coding an application, which is essentially a distributed workflow. As such, it encompasses workflow orchestration together with several types of analytics operations that can be performed at varying scale and locations across the compute continuum. Some of these operations may be isolated functions, and some may be micro-workflows dedicated to implementing a particular type of analytics. The operations are using data sets from the catalogue established at the data staging layer and may generate and register new data sets as results.

5.4.1. Ray

Ray [7] is a Python orchestrator, designed to transparently scale-out Python applications over multiple cores in the same machine and multiple machines in the same cluster. It integrates well with other technologies in EXTRACT such as Lithops and PyTorch. Ray has built-in support for workflows and ML operations of training/serving and is considered a leader in its field. A more detailed introduction to Ray is found in D3.1.

Requirement ID(s)	Justification
DM_1, DM_2	Ray supports integration with almost all 3rd-party big-data (e.g., Numpy, Pandas, Modin, Lithops, etc) and Machine Learning (PyTorch, TensorFlow, XGBoost, etc), which enables EXTRACT support for these computations in the current and future use-cases, using and managing both CPUs, GPUs, and any user-defined resources.
DM_4	Ray Data has support for dataset processing that can be customized, and (of course) we can employ regular Python code for invoking the catalogue and metadata services as needed.
DM_5	The existing Ray annotations can be extended, e.g., for supporting user-defined resources. New annotations for constraints can be added as needed.

5.4.2. COMPSs

COMP Superscalar (COMPSs) [8] is a task-based parallel programming model which aims to ease the development of applications for distributed infrastructures, such as large HPC, clouds and container managed clusters. COMPSs provides a programming interface for the development of the applications and a runtime system that exploits the inherent parallelism of applications at execution time. A more detailed introduction to COMPSs is found in D3.1.

In the following table, we describe how COMPSs addresses the Data Mining Framework requirements identified:

Requirement ID(s)	Justification
DM_1	COMPSs provides support for big data computation through its parallelization capabilities. While COMPSs is not specifically designed for Map-Reduce operations, it offers a programming model and runtime system that allows for the efficient parallel execution of data-intensive tasks. By leveraging COMPSs, we can parallelize tasks' computations and distribute them across multiple computing resources, enabling the processing of large volumes of data.
DM_3	COMPSs, as an application-level orchestrator, excels in heterogeneous environments, managing execution of complex distributed applications on a mix of hardware and locations. Thus, it can be utilized to parallelize and manage the overall workflows of EXTRACT across the compute continuum, and specifically those of serving ML models. It can handle the coordination and scheduling of tasks, ensuring efficient resource utilization and load balancing. COMPSs can therefore orchestrate the deployment and execution of the serving

	pipeline, managing the communication and synchronization between different components involved in the serving process.
DM_4	COMPSs is well-suited for integrating with data staging and facilitating seamless integration with the catalogue and metadata generation for the datasets. As the utilization of COMPSs consists of code annotations like decorators in common languages such as Python, Java or C/C++, it does not hinder the management of data movement and staging within the computational workflow.
DM_5	The existing Ray annotations can be extended, e.g., for supporting user-defined resources. New annotations for constraints can be added as needed.

5.4.3. ModelMesh

ModelMesh [9] is SoTA for cluster-level model serving in K8s, designed and proven for large-scale serving. It has also been demonstrated for small deployments, so fitting edge deployments. As K8s is a key foundation in EXTRACT, this is a good choice for model serving. It is also highly mature and robust, exceeding the TRL requirements.

ModelMesh implements an intelligent cluster-scale serving of multiple models, based on a combination of cache LRU (Least-Recently Used) of models combined with load-balancing of inference work on the cluster nodes and dynamic recovery from failures such as model loading failures and node failures. Each ModelMesh pod wraps a concrete model server, such as Nvidia Triton, Seldon MLServer, Intel OpenVino etc. ModelMesh further adds a "mesh" container in the pod that manages the co-located server while peering with other ModelMesh pods across the cluster. Last, a "puller" component in the pod allows retrieving models for serving as data from S3-compliant object storage. In that aspect, ModelMesh is well-aligned with object storage being the designated data back-bone of EXTRACT.

Requirement ID(s)	Justification
DM_3	ModelMesh allows efficient and scalable serving of multiple models in almost all available formats over Kubernetes clusters.

5.5. Data security and privacy

The techniques identified for the implementation of this task are based on three main concepts Differential Privacy, Homomorphic Encryption and Multi-Party Computation. Although the exact definition of the implementation can be done only when the design of the whole EXTRACT ecosystem is in a more advanced stage, some techniques have already been suggested especially for MPC and HE.

5.5.1. Data Privacy

The solutions identified to guarantee the maintenance of privacy of processed data are mainly based on the application of obfuscation techniques, and in particular the application of Differential Privacy (DP). DP is a system provides a rigorous mathematical definition of privacy. In the simplest setting, consider an algorithm that analyses a dataset and computes statistics about it (such as the data's mean, variance, median, mode, etc.). Such an algorithm is said to be differentially private if by looking at the output, one cannot tell whether any individual's data was included in the original dataset or not. In other words, the guarantee of a differentially private algorithm is that its behaviour hardly changes when a single individual joins or leaves the dataset. This gives a formal guarantee that individual-level information about participants in the database is not leaked by allowing data to be analysed without revealing sensitive information about any individual in the dataset.

Requirement ID(s)	Justification
SP_08	Differential Privacy will provide protection of data privacy, in order to prevent the dissemination of sensitive information. Possible Implementations: PyDP, OpenDP, PySyft.
SP_01, SP_02, SP_03	These requirements are needed to choose the most efficient implementation of DP

5.5.2. Data Security

In terms of maintaining data integrity, the selected solutions act by applying encryption algorithms to the information. In particular, the use of Fully Homomorphic Encryption (FHE): FHE is a form of encryption that allows computations to be performed on encrypted data without first having to decrypt it. FHE thus allows arbitrary computations to be performed on ciphertext, i.e., a text unreadable until it has been converted into plain text (decrypted) with a key, producing an encrypted result that can be decrypted to match the result of the same computations performed on the plaintext. FHE allows for sensitive data to be processed and analysed without the need to decrypt it first.

Since the en/de-cryption of data could be quite heavy, especially for high quantity of data, performance tests will be carried out in order to provide indications for the use of this technique also according to the overall performance required by the system.

Requirement ID(s)	Justification
SP_07	Fully Homomorphic Encryption will provide protection of the data integrity and confidentiality, in order to avoid information tampering and unauthorized access. Possible Implementations: HElib, SEAL, PALISADE
SP_01, SP_02, SP_03	These requirements are needed to choose the most efficient implementation of FHE
SP_05, SP_06, SP_07	These requirements are needed to implement the FHE

5.5.3. Process and Model Protection

The proposed security solutions will also deal with the improvement of the security of the Machine Learning models, and applications, against the adversarial threats. The main threats to be tackled are:

- **Poisoning**: Poisoning attacks involve injecting malicious data into the training dataset in order to cause the ML model to make incorrect predictions. This can be done by an attacker who has access to the training data or by an attacker who is able to influence the data that is collected by the ML system.
- **Inference**: Inference attacks involve inferring sensitive information about an individual or group based on the predictions made by an ML model. This can be done by an attacker who has access to the predictions made by the model, or by an attacker who can observe the behavior of the ML system.
- **Adversarial examples**: Adversarial examples are inputs that have been manipulated to cause an ML model to make incorrect predictions. They can be crafted to mislead the model, by adding small perturbations to the input that are imperceptible to humans but cause the model to fail.
- **Model extraction**: Model extraction attacks involve stealing the parameters of a trained ML model, which can be used to replicate the model or to attack other systems that use the same model.
- **Model inversion:** Model inversion attacks involve inferring sensitive information about the training data based on the predictions made by an ML model.
- **Privacy attacks:** Privacy attacks are a type of attack in which an attacker tries to infer sensitive information about a person by analyzing patterns in the data used to train an ML model.

Requirement ID(s)	Justification
SP_09	Multi-Party Computation will provide protection of the model integrity and confidentiality, in order to avoid the previously mentioned threads. Possible Implementations: PySyft, SecureNN, PyMPC, MPyC, SecureML
SP_05, SP_06, SP_07	These requirements are needed to implement the MPC
SP_10	The performance requirement is needed to implement the MPC in the most efficient way

6. Conclusions

The deliverable provides the result of the requirements elicitation and selection of the technologies proposed for the EXTRACT project for implementation of the data infrastructure and data mining framework layer. The deliverable focuses on several key topics, including data content and metadata layer, semantic layer requirements, cloud/edge data staging layer and interconnection with data-mining frameworks, HPC and AI & big-data frameworks, as well as data security and privacy.

In the data content and metadata layer, the requirements focus on high availability, scalability, durability, efficient resource utilization, integration with applications and services, interoperability, metadata recording, regulatory compliance and security. The proposed technology solution is Object Storage, which meets these requirements effectively.

As the technological solution for the metadata management the Nuvla platform is proposed. Nuvla offers scalability, schema-free registration, remote querying capabilities, and collaborative features, enabling efficient metadata generation, storage, querying, and sharing. The list of the capabilities matches well the corresponding requirements identified for the metadata management.

The semantic layer requirements emphasize the integration of data across disparate sources, transforming heterogeneous data into a common format, semantic search, querying and retrieval, efficient ontology support, and semantic reasoning. A hybrid architecture combining NoSQL Time Series Database (TSDB) and Graph technologies is proposed to fulfil these requirements.

In the cloud/edge data staging and integration layer, the requirements include access to storage systems, data transformation, loading data into data mining frameworks, dynamic scaling of resources based on source size, on-the-fly partitioning, and description of data format. The proposed solution focuses on data staging technologies that allow seamless integration with the catalogue and metadata generation for ingested and computed datasets.

The HPC and Big Data & AI frameworks layer requirements involve support for big data computation, machine learning model training and serving, integration with data staging, and specification of workflow and dataflow constraints. The proposed solution combines workflow orchestration (COMPSs/Ray) with various analytics operations to provide a distributed workflow and analytics capabilities.

The data security and privacy layer requirements cover data volume, velocity, variety, model structure, processing type, data confidentiality, data sensitivity, and model security. The conclusions highlight the need to consider the specific requirements for securing data and models in the EXTRACT project.

In conclusion, the proposed technologies and solutions address well the requirements of the EXTRACT project, providing a solid foundation for the implementation of the data management, metadata services, semantic layer integration, cloud/edge data staging, and data security. These technologies pave the way for effective data processing, analytics, and knowledge extraction. This will enable the project to achieve its goals of extracting valuable insights from complex and heterogeneous data sources.

7. Acronyms and Abbreviations

- EXTRACT - A distributed data-mining software platform for extreme data across the compute continuum
- HPC - High-Performance Computing
- AI - Artificial Intelligence
- WP2 - Work Package 2
- MoSCoW - Must have, Should have, Could have, and Won't have
- PER - Personalized Evacuation Route
- RL - Reinforcement Learning
- ML - Machine Learning
- TASKA - Transient Astrophysics with a Square Kilometre Array
- UnDysPuted - Unified Dynamic Spectrum Pulsar and Time Domain receiver
- RDF - Resource Description Framework
- PII - Personally Identifiable Information
- API - Application Programming Interface
- RESTful - Representational State Transfer
- SQL - Structured Query Language
- ETL - Extract, Transform, Load
- DL - Deep Learning
- ONNX - Open Neural Network Exchange
- 2FA - Two-Factor Authentication
- ACL - Access Control List
- S3 - Simple Storage Service
- TLS - Transport Layer Security
- TSDB - Time Series Database
- NoSQL - Not Only SQL
- GPL - General Public License
- DRBMS - Distributed Relational Database Management System
- ABAC - Attribute-Based Access Control
- RBAC - Role-Based Access Control
- SHP - Shapefile
- KB - Knowledge Base
- CSV - Comma-Separated Values
- HTTP - Hypertext Transfer Protocol
- CPU - Central Processing Unit
- GPU - Graphics Processing Unit
- COMPSs - COMP Superscalar
- K8s - Kubernetes
- TRL - Technology Readiness level
- LRU - Least Recently Used
- HE - Homomorphic Encryption
- MPC - Multi-Party Computation
- DP - Differential Privacy
- FHE - Fully Homomorphic Encryption

8. References

[1] A distributed data-mining software platform for extreme data across the compute continuum. https://extract-project.eu/

[2] What is MoSCoW Prioritization? | Overview of the MoSCoW Method. https://www.productplan.com/glossary/moscow-prioritization/

[3] SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/

[4] Nuvla. https://nuvla.io/ui/

[5] InfluxDB Times Series Data Platform | InfluxData. https://www.influxdata.com/

[6] J. Sampe, P. Garcia-Lopez, M. Sanchez-Artigas, G. Vernik, P. Roca-Llaberia and A. Arjona, "Toward Multicloud Access Transparency in Serverless Computing," in IEEE Software, vol. 38, no. 1, pp. 68-74, Jan.-Feb. 2021, doi: 10.1109/MS.2020.3029994.

[7] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, & Ion Stoica. (2018). Ray: A Distributed Framework for Emerging AI Applications.

[8] Badia, Rosa M. & Conejero, Javier & Diaz, Carlos & Ejarque, Jorge & Lezzi, Daniele & Lordan, Francesc & Ramon-Cortes Vilarrodona, Cristian & Sirvent, Raul. (2015). COMP Superscalar, an interoperable programming framework. SoftwareX. 3. 10.1016/j.softx.2015.10.004.

[9] ModelMesh Overview - KServe Documentation Website. https://kserve.github.io/website/0.8/modelserving/mms/modelmesh/overview/