SEcure Decentralised Intelligent Data MARKetplace

D3.4 Enabling tools for data interoperability, distributed data storage and training distributed AI models. Final version

Document Identification	
Contractual delivery date:	31/07/2025
Actual delivery date:	08/08/2025
Responsible beneficiary:	EGM
Contributing beneficiaries:	EGM, INRIA, LINKS, UC, SURREY. WINGS, NUID UCD, ATOS
Dissemination level:	PU
Version:	1.0
Status:	Final

Keywords:
Data, model, interoperability

This document is issued within the frame and for the purpose of the SEDIMARK project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No.101070074. and is also partly funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission or UKRI.

Document Information

Document Identification			
Related WP	WP3	**Related Deliverables(s):**	SEDIMARK_D3.2
Document reference:	SEDIMARK_D3.4	**Total number of pages:**	89

List of Contributors	
Name	**Partner**
Gilles Orazi, Thomas Bousselin, Franck Le Gall, Luc Gasser Léa Robert, Julien Fleury	EGM
Shahin Abdoul Soukour, Nikolaos Georgantas	INRIA
Alberto Carelli, Andrea Vesco	LINKS
Juan Ramón Santana, Pablo Sotres, Víctor González, Jorge Lanza, Luis Sánchez	UC
Tarek Elsaleh, Peipei Wu Mahrukh Awan, Sneha Hanumanthaiah, Adrian Hilton	SURREY
Grigorios Koutantos, Panagiotis Vlacheas, Dimitris Zagganas	WINGS
Diarmuid O'Reilly Morgan, Erika Duriakova, Honghui Du, Elias Tragos, Aonghus Lawlor, Neil Hurley	NUID UCD
César Caramazana Joaquín García	ATOS

Document History			
Version	**Date**	**Change editors**	**Change**
0.1	26/02/2025	EGM	Creation and initial contributions
0.2	03/05/2025	INRIA	Contribution to section 3
0.3	12/05/2025	ATOS	Updates in Fleviden sections
0.4	07/06/2025	INRIA	Contribution to section 3
0.5	06/05/2025	SURREY	Sections 2.3, 2.4, 3.8., 4.2.1, 4.2.2
0.6	06/06/2025	SURREY	Sections 2.3, 2.4, 2.6, 3.8., 4.2.1, 4.2.2, 6.2.3
0.7	25/06/2025	LINKS, NUID UCD, WINGS	Sections 4, 5
0.8	02/07/2025	UC	Section 6.2.3
0.85	11/07/2025	EGM	Finalisation for reviews
0.86	31/07/2025	EGM	Integration of technical reviews from NUID UCD and MYT
0.9	06/08/2025	ATOS	Quality Format Review
1.0	08/08/2025	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	**Who (Partner short name)**	**Approval date**
Reviewer 1	Nikolaos Babis (MYT)	15/07/2025
Reviewer 2	Elias Tragos (NUID UCD)	17/07/2025
Quality manager	María Guadalupe Rodríguez (ATOS)	08/08/2025
Project Coordinator	Miguel Ángel Esbrí (ATOS)	08/08/2025

Table of content

List of figures

List of tables

List of Acronyms

Abbreviation / Acronym	Description
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
CSV	Comma Separated Values
CWL	Common Workflow Language
DCAT	Data Catalog Vocabulary
DCT	DC (Dublin Core) Term
DLT	Distributed ledger technology
DQV	Data Quality Vocabulary
DSW	Dimension-Segment-Wise
Dx.y	Deliverable number y belonging to WP x
ETL	Extract Transform Load
ETSI	European Telecom Standards Institute
EVM	Ethereum Virtual Machine
FAIR	Findable, Accessible, Interoperable, and Reusable
FL	Federated Learning
FLOPS	Floating Point Operations Per Second
FOAF	Friend Of A Friend
FTP	File Transfer Protocol
GL	Gossip Learning
GPT	Generative Pre-trained Transformer
HED	Hierarchical Encoder-Decoder
HORNET	IOTA Full-Node Software
HTTP(S)	Hypertext Transfer Protocol (Secure)

Abbreviation / Acronym	Description
IDS	International Data Spaces
IDSA	International Data Spaces Association
IEEE	Institute of Electrical and Electronics Engineers
INX	IOTA Node Extension (INX) interface
ISC VM	IOTA Smart Contracts Virtual Machine
ISG CIM	Industry Specification Group for Context Information Management
JSON-LD	JavaScript Object Notation for Linked Data
LLM	Large Language Model
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MPC	Multi-Party Computation
MSE	Mean Squared Error
NGSI-LD	Next Generation Service Interfaces for Linked Data
ODRL	ODRL
ONNX	Open Neural Network Exchange
OWL	Web Ontology Language
POI	Proof of Inclusion
PSI-CA	Private Set Intersection Cardinality Protocol
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	Representational State Transfer
RMSE	Root Mean Squared Error
RSE	Relative Squared Error
SDM	Smart Data Models
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System

Abbreviation / Acronym	Description
SPARQL	SPARQL Protocol and RDF Query Language
SSE-C	server-side encryption with customer-provided keys
TRL	Technology Readiness Level
URI	Uniform Resource Identifier
WLAN	Wireless Local Area Network
WP	Work Package
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language

Executive Summary

This report presents the design and implementation of the core technical enablers for the SEcure Decentralised Intelligent Data MARKetplace (SEDIMARK) platform. The project aims to address the limitations of centralized data markets by fostering a secure, trusted, and intelligent ecosystem based on Distributed Ledger Technology (DLT) and Artificial Intelligence (AI). The work detailed in this document establishes the foundational components for interoperability, AI-driven services, DLT-based trust, and distributed storage, advancing the platform from a Technology Readiness Level (TRL) of 5 toward demonstration in real-world scenarios.

A key contribution of this work is a comprehensive interoperability framework. At its core, the framework uses the NGSI-LD specification to create a common semantic language for data assets. This is supplemented by a Marketplace Information Model, which defines crucial marketplace concepts such as Self-Description, Offering, and Asset. This model builds upon existing standards like DCAT and ODRL by introducing the "Offering" concept, which allows multiple diverse assets—such as datasets, AI models, and services—to be bundled and transacted together. A suite of software components within the Interoperability Enabler handles data formatting, curation, quality annotation, and validation to ensure data adheres to FAIR principles.

The platform's intelligence is powered by a multifaceted AI Enabler. This component supports advanced local model training with techniques like the transformer-based CrossFormer for multivariate time-series forecasting and model optimization methods like pruning. For collaborative scenarios, the project introduces two frameworks for distributed training: deFLight, a dynamic and fully decentralized framework supporting gossip and federated learning, and Fleviden, a tool for orchestrating complex federated workflows. A significant innovation is the Offering Generator, which uses Large Language Models (LLMs) to automatically create standards-compliant, semantically rich marketplace offerings from unstructured metadata, lowering the barrier to entry for data providers.

Trust and security are ensured by a DLT infrastructure built on a private instance of the IOTA Tangle (Layer 1) and IOTA Smart Contracts (Layer 2). This two-layer architecture provides a non-repudiable ledger for managing participant identities, cataloguing offering metadata, and facilitating secure asset trading. To support the platform's digital assets, a robust Storage Enabler provides a distributed architecture using Minio for AI model storage and NGSI-LD brokers for scalable, interoperable storage of linked data.

1 Introduction

1.1 Purpose of the document

This document details the design and implementation of the core technical enablers for the SEDIMARK platform. These components are fundamental to the project's goal of enabling seamless, secure, and intelligent data sharing among diverse participants. The work focuses on four key pillars: comprehensive interoperability, advanced AI capabilities, a trust-based DLT infrastructure, and scalable distributed storage.

A central achievement is establishing interoperability at multiple levels through:

- **Standardized Information Models**: The adoption of NGSI-LD as a base format for data entities creates a common semantic language for all assets.
- **Marketplace Ontology**: A Marketplace Information Model defines the core concepts of Self-Description, Offering, and Asset, providing a structured framework for participants to register, discover, and exchange resources.
- **Practical Interoperability Tools**: A suite of components, known as the "Interoperability Enabler," provides functionalities for data formatting, curation, quality annotation, and validation, ensuring data conforms to SEDIMARK standards.

The AI Enabler offers sophisticated tools for both local and collaborative machine learning, including:

- **Advanced Local Training**: Support for training complex models like the transformer-based CrossFormer for time-series forecasting and advanced optimization techniques such as model pruning.
- **Distributed Learning Frameworks**: The document presents two distinct frameworks for distributed training: deFLight, a dynamic, decentralized framework, and Fleviden, an extensible tool for orchestrating federated learning workflows.
- **Automated Offering Generation**: An innovative component that uses Large Language Models (LLMs) to automatically generate semantically rich, standards-compliant marketplace offerings from unstructured metadata.

Underpinning these capabilities are robust infrastructure components:

- **DLT Infrastructure**: A private DLT instance using IOTA Tangle (Layer 1) and IOTA Smart Contracts (Layer 2) establishes a trustworthy and immutable ledger.
- **Storage Enabler**: SEDIMARK utilizes a distributed storage architecture, including Minio for AI models and NGSI-LD brokers for scalable, interoperable data storage.

1.2 Relation to another project work

This deliverable presents the work related to the components meant to support interoperability, distributed storage and system intelligence. These are three main pillars within SEDIMARK to support the project objectives for enabling seamless data sharing between consumers and providers. Interoperability is a key part of SEDIMARK to enable the efficient and easy reuse of datasets, models and services across the whole network of SEDIMARK participants, aiming i.e. to support them in integrating data from different sources to train more advanced and robust models or to enable the distributed training of machine learning models on compatible

datasets. This report is an update of the SEDIMARK Deliverable D3.3 and reports over the progresses made in WP3.

1.3 Structure of the document

Figure 1 presents the SEDIMARK functional architecture. The components highlighted in orange are detailed in this deliverable, bridging the data, intelligence, and service layers of the platform. The document is structured as follows1:

- **Section 2: Interoperability Assets**: Describes the information models for data (NGSI-LD) and marketplace concepts (Offerings, Assets), which are fundamental to achieving interoperability.
- **Section 3: The Interoperability Enabler**: Details the software components responsible for data formatting, curation, quality annotation, and validation.
- **Section 4: The AI Enabler**: Presents the frameworks for local and distributed model training (CrossFormer, deFLight, Fleviden), model optimization, and the LLM-based Offering Generator.
- **Section 5: The DLT Infrastructure**: Explains the architecture and software stack of the IOTA-based distributed ledger used for trust and transactions.
- **Section 6: The Storage Enabler:** Outlines the distributed storage solutions for data and AI models within the SEDIMARK ecosystem.

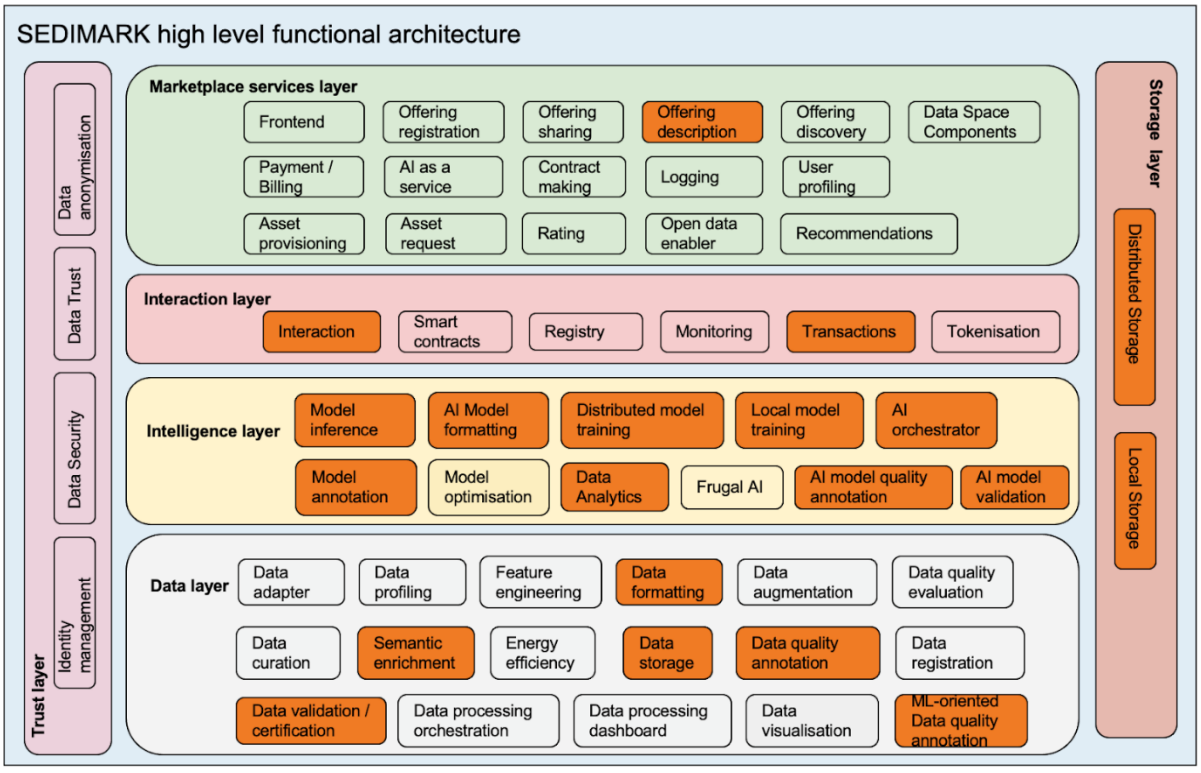


Figure 1: SEDIMARK Functional Architecture: orange highlights functional components that are being part of this deliverable

2 Interoperability assets

2.1 Introduction

Interoperability is a crucial facet of modern information management, enabling seamless communication and exchange of data, models and services across diverse systems, platforms, and applications. In a world characterized by an abundance of data sources and formats, achieving interoperability ensures that disparate systems can understand, interpret, and effectively use shared data. This capability facilitates collaboration, integration, and synergy among organizations and technologies, breaking down silos and promoting a more interconnected digital ecosystem.

This introduction explores the significance of data interoperability in overcoming the challenges posed by data heterogeneity, promoting standardization, and ultimately unlocking the full potential of interconnected data landscapes. Embracing data interoperability not only enhances operational efficiency but also lays the groundwork for advanced analytics, artificial intelligence, and the seamless flow of information in our interconnected, data-driven world.

Before initiating any data processing pipeline within the SEDIMARK platform, data thus need to be formatted so to be usable by the pipeline. In its initial version, it has been agreed that the data processing pipeline would consumes and produces data organised along the NGSI-LD information model [1].

2.2 Assets information models

2.2.1 NGSI-LD as base format

NGSI-LD is represented in JSON-LD and thus have a RDF grounding. It is mainly based on RDF standards to capture high-level relations between entities (representing or not a real-world asset) and properties of entities, as shown below. The core concept in the NGSI-LD data model is the "*Entity*" which can have properties and relationships to other entities. An *Entity* is equivalent to an OWL class. The assumption is that the world consists of entities, which can be physical entities like a car or a building, but also more abstract entities like a company or the coverage area of WLAN access points. *Entity* instances are identified by a unique URI and a type, e.g., a sensor with identifier urn:ngsi-ld:Sensor:01 and of type Sensor. Different from rdf:Properties, NGSI-LD properties (and relationship) are also considered as OWL classes also. Properties and relationships can be annotated by properties and relationships themselves, e.g. a timestamp, the provenance of the information or the quality of the information can be provided. The hasObject and hasValuein the NGSI-LD metamodel are defined to enable RDF reification, based on the blank node pattern, to leverage the property graph model.

The NGSI-LD cross-domain ontology extends the NGSI-LD metamodel to cover several general contexts presented below [2]:

- Mobility defines the stationary, movable or mobile characteristics of an *Entity*;
- Location differentiates and provides concepts to model the coordination based, set based or graph-based location;
- Temporal specification includes property and values for temporal property definitions;

- Behavioural system includes properties and values to describe system state, measurement and reliability;
- System composition and grouping provides a way to model system of systems in which small systems are composed together to form a complex system following specific patterns.

The NGSI-LD cross domain ontology is presented in Figure 2.

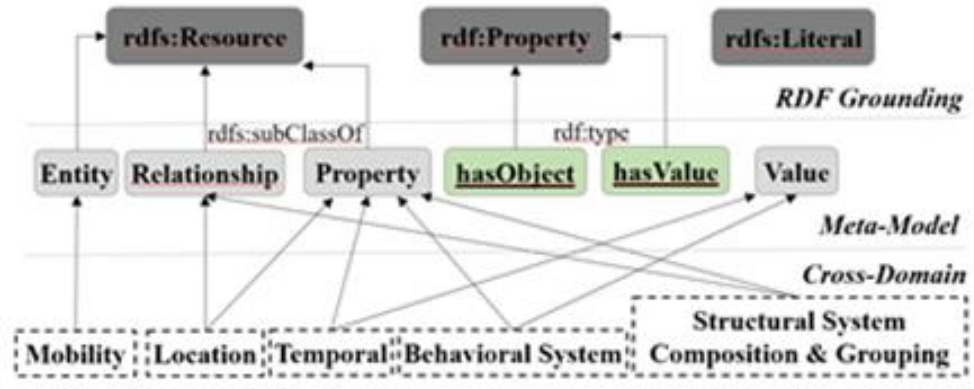


Figure 2: RDF standards to capture high-level relations between entities in NGSI-LD (source ETSI [1])

Below we present a use case example for modelling data and context using the NGSI-LD. The example consists of a station that returns the measure of the level and flow of a river. This station has an id which is urn:ngsi-ld:Hydrometric-Station:X061000201. This station is located in a river identified by urn:ngsi-ld:River:La_Durance. This is defined by the relationship (isLocatedOn).

To model this example, Figure 3 presents the main symbols signification used in the medialisation task.

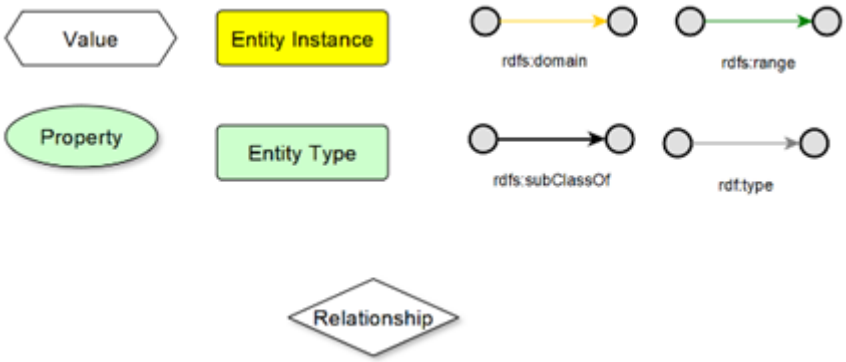


Figure 3: Main Symbols Definition (source ETSI [1])

The *Entity* "River" (since it is a subclass of NGSI-LD Entity) is instantiated with the identifier urn:ngsi-ld:River:La_Durance. Several relationships are defined in this example: the first (isAffluentOf) describes the hierarchy between the rivers, to be used later on for graph-based data processing. The relationship hasWeatherInformation provides weather related information for the river.

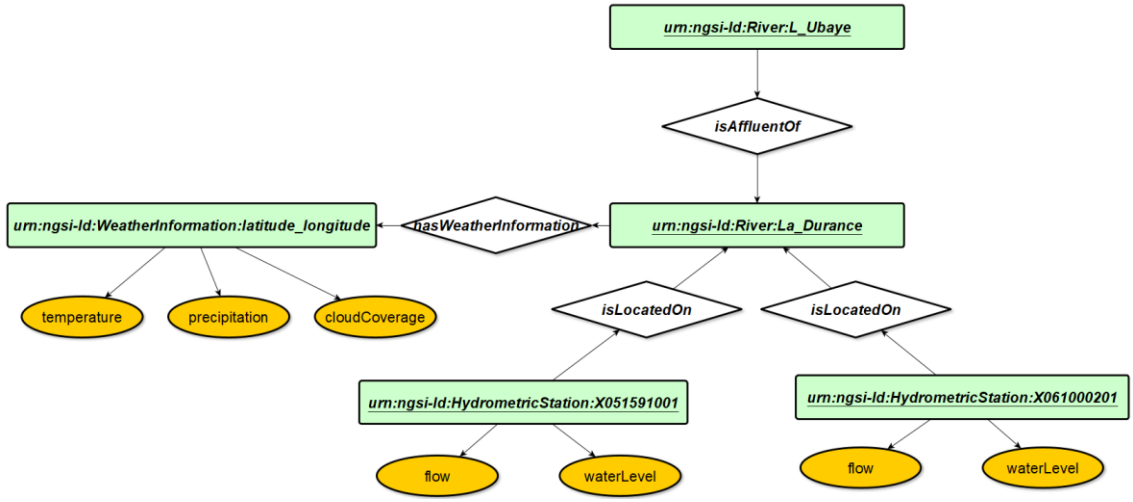


Figure 4: Illustration of NGDI-LD usage (extracted from water use case)

2.2.2 NGSI-LD Smart Data Models

The Smart Data Models (SDM) initiative, aims to offer a standardized approach to data representation across different domains. It aims to enhance interoperability between diverse systems and applications, thus enabling seamless communication. Developed by the FIWARE community, the Smart Data Models are open source and are developed through constant efforts from the community.

Within SEDIMARK, implementing Smart Data Models for Data Assets aims to establish a homogeneous approach for participants to utilize the reusable common tools proposed by SEDIMARK, including AI modelling and data processing. Therefore, it complements the like NGSI-LD semantic-enabled APIs with NGSI-LD data models. The implementation of Smart Data Models ensures that providers reveal a consistent taxonomy. This enables SEDIMARK participants to both sell enhanced data and expand the pool of potential customers and data providers within the Marketplace for service providers.

Smart Data Models offer a customisable framework suitable for diverse domains, allowing for the creation of multiple domain-specific data models that cater to applications or datasets.

SEDIMARK advocates for the practical use of Smart Data Models in Data Assets, despite the possibility of needing to adjust proposed models with new attributes and properties. Several data models have been identified from the domains supported by the initiative, including *Smart Mobility*, *Smart Cities Smart Environment* and *Smart Energy*. Additionally, Smart Data Models, such as the Data Quality model, can be used to enrich the content of existing datasets with the output of the data processing pipeline.

2.2.3 NGSI-LD API

2.2.3.1 Introduction

The NGSI-LD API supports several operations, with messages expressed in JSON-LD. The API is the standard for management of context information (which can be summarised as being any piece of information associated with a context such as time-location information). The overall NGSI-LD API operations include:

Table 1: NGSI-LD operations

General Operations	Batch Operations
Entity create	Batch Entity Creation
Entity update	Batch Entity Create/Update (Upsert)
Entity partial update	Batch Entity Update
Entity delete	Batch Entity Delete
Entity retrieval	
Queries	**Temporal Operations**
Subscriptions	Create/Update Temporal Entity Representation
Registry Operations	Add Attributes to Temporal Entity Rep.
CSRegistryEntry create	Delete Attribute from Temporal Entity Rep.
CSRegistryEntry update	Modify Attribute Instance in Temporal Entity Rep.
CSRegistryEntry partial update	Delete Attribute Instance from Temporal Entity Rep.
CSRegistryEntry delete	Delete Temporal Entity Representation
CSRegistryEntry retrieval	Retrieve Temporal Entity Evolution
CSRegistryEntry query	Query Temporal Entity Evolution
CSRegistryEntry subscription	

This API relies on the NGSI-LD data model introduced earlier. In short, this model makes use of the JSON-LD serialisation format which adds linked data capabilities to the JSON format. The core of the model builds upon the concept of Entity, where an entity can have Properties and Relationships with other entities, building a property graph model.

The JSON-LD format allows to create a network of standards-based machine interpretable data across different sources. The JSON-LD format includes an @context clause used to map short terms used in the serialization to URIs uniquely identifying concepts and mapping to specific types (e.g. *DateTime*).

In the following, we present the modelling process of the previous example using the NGSI-LD API based on JSON-LD messages for creating and querying instances of Sensor and Station.

2.2.3.2 Creating an instance of Entity

An *Entity* can be created using the following endpoint (among others):

POST {gatewayServer}/ngsi-ld/v1/entities

The payload must contain at least an id and a type for the entity. Any other attribute can also be added to the entity when creating it.

An example of payload used for the creation of a hydrometric station entity for the water use case is given below:

```
{
"id": "urn:ngsi-ld:HydrometricStation:X031001001”,
"type": "HydrometricStation”,
"location": {
            "type": "GeoProperty",
            "value": {
                "type": "Point",
                "coordinates": [
                6.2727640,
                44.4709131
            ]
        }
    }
}
```

Figure 5: example of NGSI-LD payload

2.2.3.3 Creating an instance of an attribute in an Entity

An instance of an attribute can be added to an *Entity* using the following endpoint (among others):

PATCH {gatewayServer}/ngsi-ld/v1/entities/urn:ngsi-ld:HydrometricStation:X031001001

The payload can contain an instance for any attribute (already existing or not), if an attribute does not exist, it will be created with the new instance.

An example of payload used to add some flow and water level measurements to a hydrometric station for the water use case is given below

```
{
      "flow": {
          “value" : 138000.0,
          "observedAt" : "2023-12-04T10:15:00Z",
          "type" : "Property",
          "unitCode" : "G51"
      },
      "waterLevel": {
          "value" : 1237.0,
          "observedAt" : "2023-12-04T10:15:00Z",
          "type" : "Property",
          "unitCode" : "MMT"
      }
}
```

Figure 6: Payload to add instance of attributes to an Entity.

2.2.3.4 Retrieving an Entity by Id query

An *Entity* can be retrieved using the following endpoint (among others):

GET {{gatewayServer}}/ngsi-ld/v1/entities/{{entity_id}}

An example of the response given for the entity used in the previous example is given below:

```
{
    "id": "urn:ngsi-ld:HydrometricStation:X031001001",
    "type": "HydrometricStation",
    "flow": {
        "type": "Property",
        "value": 139000.0,
        "observedAt": "2023-12-04T07:45:00Z",
        "unitCode": "G51"
    },
    "waterLevel": {
        "type": "Property",
        "value": 1238.0,
        "observedAt": "2023-12-04T07:45:00Z",
        "unitCode": "MMT"
    },
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [
                6.49800996,
                44.55535641
            ]
        }
    }
}
```


Figure 7: Response given for an Entity request.

Figure 7 show the current state of the Entity (i.e., only the last instances for each attribute are displayed).

The history of the *Entity* can be retrieved using this endpoint (among others):

{{gatewayServer}}/ngsi-ld/v1/temporal/entities/{{entity_id}}?timerel=after&timeAt={{datetime}}&options=temporalValues

An example of the response given for the entity used in the previous example is given below:

```
{
    "id": "urn:ngsi-ld:HydrometricStation:X031001001",
    "type": "HydrometricStation",
    "flow": {
        "type": "Property",
        "values": [
            [
                80500.0,
                "2023-12-01T00:15:00Z"
            ],
            [
                82800.0,
                "2023-12-01T00:30:00Z"
            ],
            [
                85100.0,
                "2023-12-01T00:45:00Z"
            ], …
      }, …
}
```


Figure 8: Retrieving history (timeseries) of an Entity attribute.

2.3 Marketplace information models

The Marketplace Information Model is an RDFS/OWL-ontology covering the fundamental concepts of SEDIMARK needed for the registration of Participants and the discovery and exchange of Offerings and Assets. This model establishes a common framework to ensure interoperability within a SEDIMARK-based Marketplace and includes the terms defined in Deliverable SEDIMARK_D2.3 [3] to enable participants to discover and exchange Assets in the form of Offerings. This common ontology is meant to serve as a shared language, fostering seamless communication and interoperability among the users of SEDIMARK. Therefore, the use of this information model is enforced for any Participant or component that wants to join the Marketplace based on SEDIMARK guidelines. The main goal of this model is to ease the search and discovery of Participants and their offers, describing accurately their information.

The creation of this model is supported by existing proposals by similar initiatives and is built upon well-known ontologies such as Open Digital Rights Language (ODRL) [4], Data Catalog vocabulary (DCAT) [5], Friend Of A Friend (FOAF) [6] or the Dublin Core Terms (DCT) [7]. In particular, the model has its foundations in the proposal shared by the International Data Spaces Protocol [8], to align as much as possible with such an initiative, although including

new terms introduced by SEDIMARK (e.g., the concept of Offering; the additional type of Assets that can be part of the Marketplace; or the data quality information that is part of SEDIMARK).

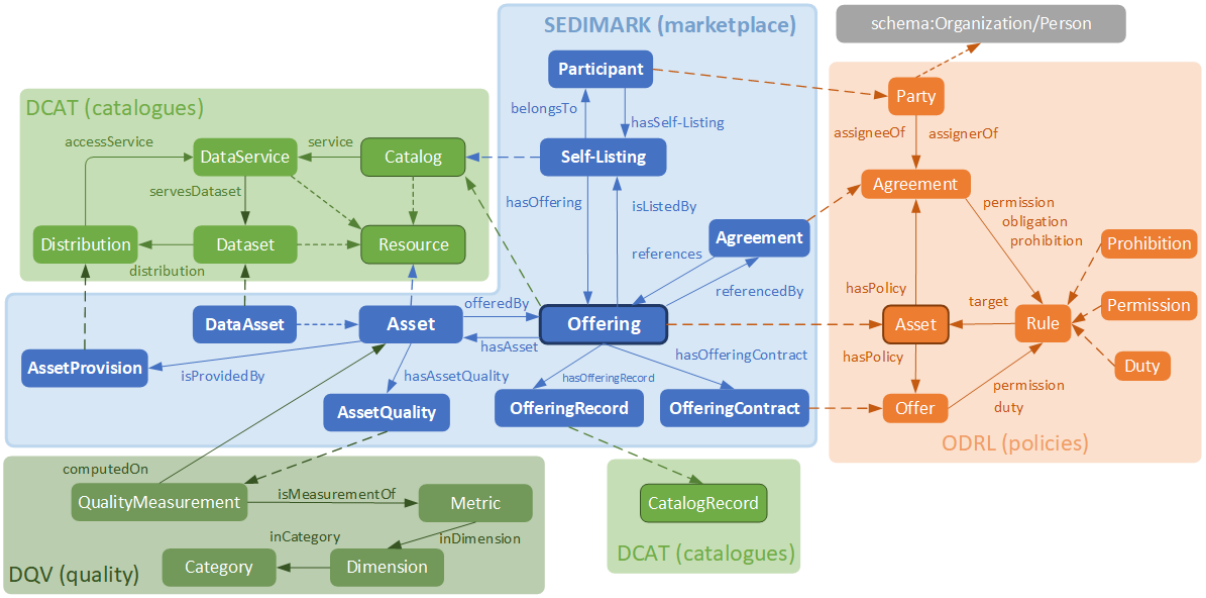


Figure 9: High-level view of the Marketplace Information Model

The current version of the Marketplace Information Model is depicted in Figure 9. The ontology has been designed using the Protégé tool [9] documented using the WIDOCO tool and hosted online via GitHub and GitHub Pages. The main concepts in this information model are the Self-Description, Participant, Self-Listing, Offering and Asset and Asset Provision. Figure 10 illustrates the properties for each of these main concepts.

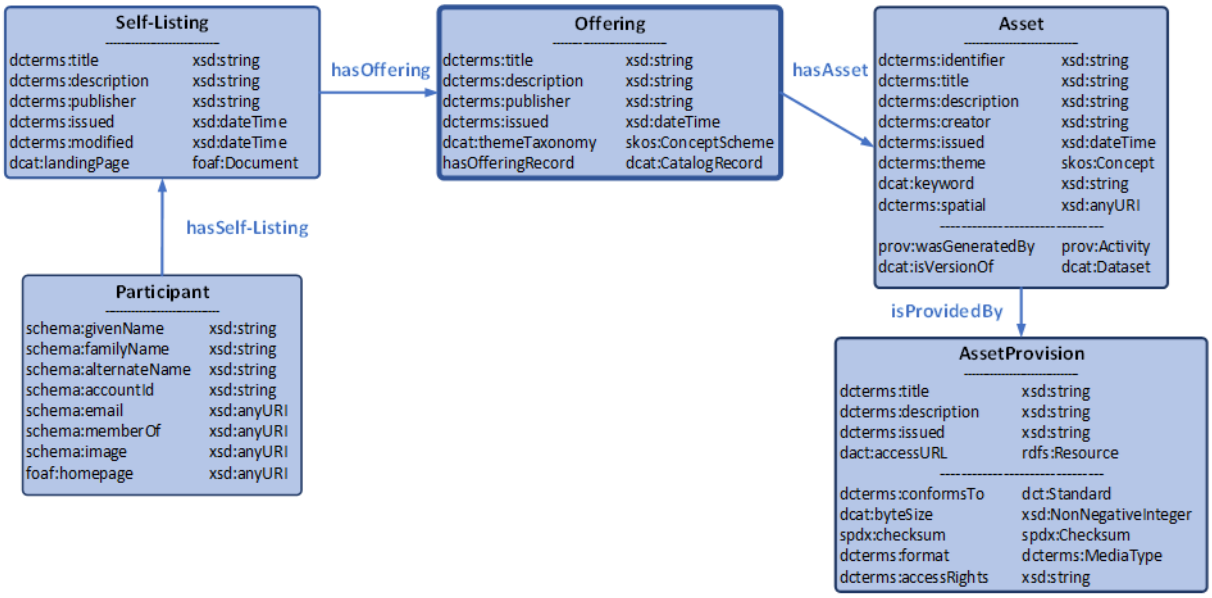


Figure 10: Properties for main classes in the Marketplace Information Model

2.3.1 Self-Description

As defined in Deliverable SEDIMARK_D2.3 [3], Self-Description is a machine-interpretable document providing all the information about a Participant. In this case, it can be considered the main class within the Marketplace Information Model. This concept is also a core part of other information models, such as the ones from Gaia-X [10] and IDS [11]. Any Participant in a Marketplace must provide a Self-Description.

There are several concepts that are part of the Self-Description, including the information about the Participant (name, description and the timestamps where this information was updated or created). Besides, the Self-Description can also link to a Self-Listing concept, which lists the set of Offerings from a Participant acting as a Provider.

```
{
    "@id": "https://connector.eu/",
    "@type": "sedimark:self-description",
    "dct:issued": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "dct:modified": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "foaf:name": "SEDIMARK Participant A",
    "dct:description": "Participant located in Europe...",
    "dct:language": {
        "@id":
"http://publications.europa.eu/resource/authority/language/ENG"
    },
    "sedimark:hasSelf-listing": {
        "@id": "https://connector.eu/self-listing"
    },
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-
model/0.1/"
    }
}
```

Figure 11: Self-Description JSON-LD example

2.3.2 Offering

The Offering is a concept introduced by SEDIMARK and describes and bundles a set of Assets that are part of an offer, along with their terms and conditions. This concept is conceived essentially as a subclass of a DCAT *Catalog* (as well as the Self-Listing concept) with

additional properties to link to other SEDIMARK concepts such as Assets and Offering Contracts. As mentioned, only Participants acting as a Provider has a Self-Listing along with a set of Offerings.

The Offering concept is also a key difference between the SEDIMARK Marketplace Information Model and the IDS Protocol [11]. In the IDS protocol, every offer is composed of a single Asset, while in SEDIMARK they can be grouped in an Offering, thus containing multiple assets per transaction.

Finally, one important aspect of Offerings is contracting. Each Offering contains a mandatory OfferingContract object and, possibly, an Agreement. Both concepts, Contract and Agreement, are subclasses of ODRL Offer and Agreement concepts, respectively. While a single Contract object is mandatory (even if there are no particular restrictions) in each Offering, Agreement objects are only required per transaction. Agreements are similar to contracts but add specific properties (i.e., assigner and assignee), which specify the Participants tied to the policies that are part of the Offering Agreement.

```
{
    "@type": "sedimark:self-listing",
    "@id": "https://connector.eu/self-listing",
    "sedimark:belongsTo": {
        "@id": "https://connector.eu/"
    },
    "sedimark:hasOffering": [
        {
            "@id": "https://connector.eu/offering/offeringID"
        }
    ],
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-
model/0.1/"
    }
}
```

Figure 12: Self-Listing JSON-LD example

```
{
    "@id": "https://connector.eu/offering/offeringID",
    "@type": "sedimark:OfferingContract",
    "sedimark:participantId": "https://connector.com/",
    "dct:title": "offeringName",
    "dct:description": "University from the North of Spain...",
    "dct:issued": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "dct:modified": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "dcat:keyword": [
        "keyword 1",
        "keyword 2"
    ],
    "odrl:hasPolicy": {
        "@id": "https://connector.eu/policy/policyID",
        "@type": "sedimark:Contract",
        "odrl:permission": [],
        "odrl:prohibition": [],
        "odrl:obligation": []
    },
    "sedimark:hasAsset": [
        {
            "@id": "https://connector.eu/asset/assetID"
        }
    ],
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-
model/0.1/"
    }
}
```


Figure 13: Offering JSON-LD example

2.3.3 Asset

Assets are the resources being offered in each of the Offerings. Initially, three different Asset concepts were considered in SEDIMARK depending on the type of the resource they are representing. In this sense, the Assets defined are datasets (either static or streaming data), AI Models, Services (e.g., data processing) and other Assets, such as containers or virtual

machines. This is also another difference with the IDS Protocol proposal, as assets are only related to data, either streaming or static datasets.

In addition, another concept has been defined within the Marketplace Information Model to represent the quality of the Asset which extends the QualityMeasurement concept from the Data Quality Vocabulary (DQV) ontology, thus giving an idea of the Asset composing an Offering, to foster the exchange and represent what SEDIMARK tools through the Data Processing Pipeline can provide as an added value to providers which enhance their data through SEDIMARK.

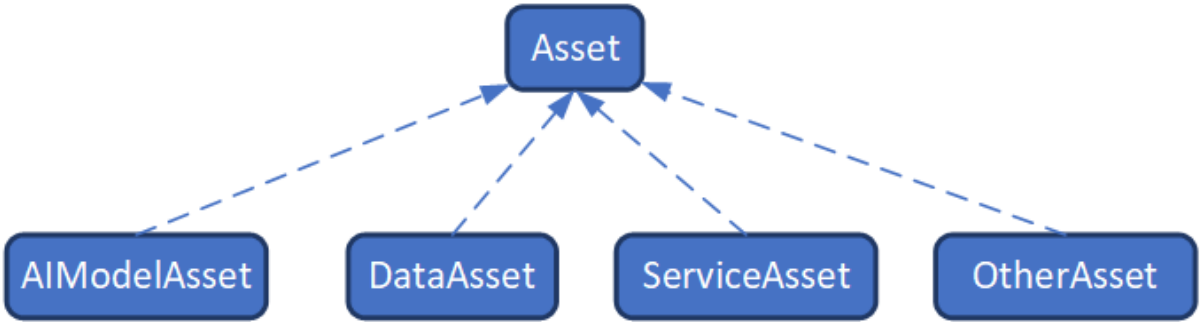


Figure 14: Types of Assets in the Marketplace Information Model

Data Assets

Datasets are represented in the Marketplace Information Model as a subclass of the *Dataset* class from the DCAT ontology. In turn, the DataAsset includes properties relating to descriptions, keywords, and spatial, temporal and thematic contexts.

```
{
    "@id": "https://connector.eu/asset/assetID",
    "@type": "sedimark:Dataset",
    "dct:description": "data asset description",
    "dct:language": {
        "@id": "http://publications.europa.eu/resource/authority/language/ENG"
    },
    "sedimark:hasDataQuality": {
        "@type": "sedimark:dataQuality",
        "@id": "https://connector.eu/dataquality/dataQualityID"
    },
    "dcat:distribution": [{
        "@id": "https://connector.eu/dataquality/distributionID",
        "@type": "dcat:Distribution",
        "dct:format": {
            "@id": "HttpProxy"
        },
        "dct:issued": {
            "@type": "xsd:dateTime",
            "@value": "2023-11-06T16:54:48.577964"
        },
        "dct:modified": {
            "@type": "xsd:dateTime",
            "@value": "2023-11-06T16:54:48.577964"
        },
        "dcat:mediaType": {
            "@id": "https://www.iana.org/assignments/media-types/application/ld+json"
        },
        "dcat:accessService": {
            "@id": "https://connector.eu/serviceID",
            "@type": "dcat:DataService",
            "dcat:endpointDescription": "NGSI-LD API",
            "dcat:endpointURL": {
                "@id": "https://connector.eu/assetID/protocol"
            }
        }
    }],
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-model/0.1/"
    }
}
```


Figure 15: DataAsset JSON-LD example

AI Model Assets

AI Model Assets represent exchangeable AI Models for both centralised and distributed AI learning techniques. Therefore, the AI Model Asset reflects a number of aspects of AI models which will be exploited by search and discovery mechanisms to retrieve relevant offerings for Consumers. The following properties have been identified for this class:

Table 2: Properties for the AI Model Asset

Property	Description
category	Type of machine learning algorithms, i.e. Supervised Learning, Unsupervised Learning, Semi-supervised learning, Reinforcement Learning
purpose	General purpose of the Model e.g. Classification, Natural Language Understanding, Recommendation, Forecasting, Synthetic Data Generation etc.
algorithm	The algorithm used for the model, e.g. Neural Network,
serialization	The model serving serialisation format; e.g. TensorFlow, parquet, PyTorch etc.
version	The version of the Model. This is particularly important for decentralised learning
execution	How the model will be deployed (parallel execution of the algorithms) (centralised, federated etc,)
size	memory size of the model in Gigabytes
modified	when the model was last modified
handleStream	whether the model is adapted to work with stream data
inputFormat	Accepted input format for the model
inputParameters	Parameters passed to the model (Stringified array)
outputFormat	Output format for the model
outputParameters	Parameters passed by the model (Stringified array)
hasTrainingDataset	The dataset used for training the AI model Asset
hasArchitecture	The network architecture that the model adopts

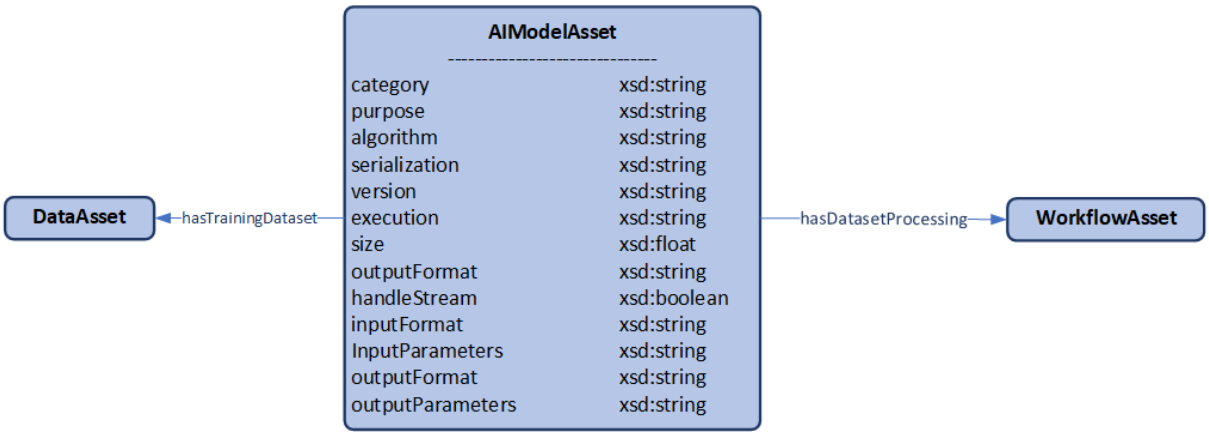


Figure 16: AI Model Asset Properties

ServiceAssets

The ServiceAsset class covers service-based assets, such as the provision of data storage or computation resources, or the application of an AI Model Asset on a Data Asset(s). Each of these assets represents the information describing the particularities of each service (e.g., number and type of processor cores, storage type, etc.).

OtherAssets

Other type of assets, such as Virtual Machines or Containers are included here, which will include additional properties to define their characteristics (e.g., computation requirements, operating system, etc.).

WorkflowAssets

This Asset reflects the artefact that defines a workflow for either data processing or AI Model Asset generation or use for inference. Section 2.5 provides details on how the Asset is generated.

2.4 Vocabularies alignment

For the Assets to reflect themes or use cases, a vocabulary is required to provide a unified reference for naming aspects relating to information captured, which are represented as properties that correspond to the use case's domain of interest.

2.4.1 Theme Vocabulary

Assets provided within SEDIMARK marketplace are associated with real and virtual entities of interest, such as vehicles or weather stations, that are related to a particular theme or use case, such as “transportation” or “environment”. To capture this, a vocabulary has been established that reflects the theme taxonomy adopted by the Smart Data Models initiative. The vocabulary is essentially comprised of instantiations of the Concept class that belongs to the SKOS Ontology [12], which links with the SEDIMARK ontology via the *dcat:theme* property, as part of the *Asset* class. The vocabulary as a whole is also explicitly declared through instantiating the *ConceptScheme* class, which links with the SEDIMARK ontology via the dcat:themeTaxonomy object property, as part of the Offering class.

The main or top concepts of the vocabulary are the *Domain* of interest, the *Subject* or sub-domain, the *Entity* of interest and *Properties* associated with that *Entity*. The main concepts extend the SKOS ontology for knowledge organization. Relational properties from the SKOS ontology are used to link *Entities* with their Properties, Entities with their *Subjects*, and *Subjects* with their Domains. Hierarchical properties from SKOS are used to link specific *Entity* concepts, such as Transportation with its skos:broader concept, i.e. Entity.

The instance naming convention follows the naming scheme adopted by SmartDataModels.

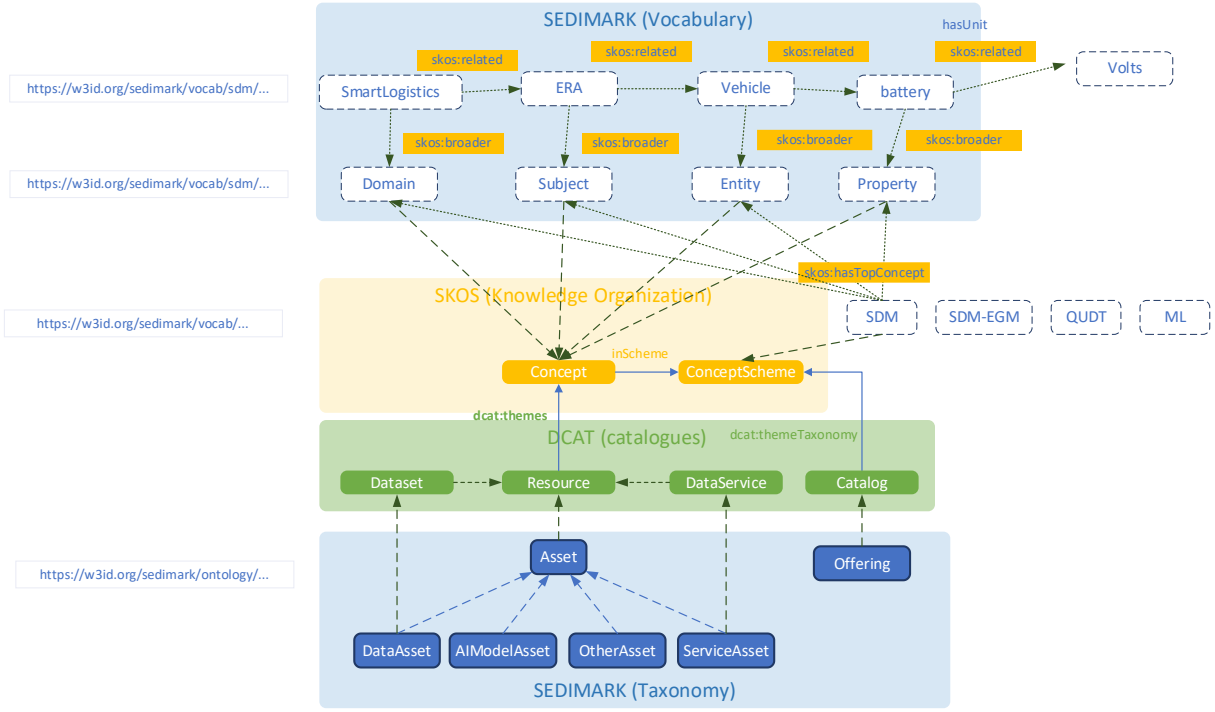


Figure 17: Vocabulary for the SEDIMARK Ontology

2.4.2 Vocabulary publication and sharing

The vocabulary described in Section 2.4.1 is made available in a machine-readable format using JSON-LD (JavaScript Object Notation for Linked Data). JSON-LD is a W3C-recommended specification [21] designed to enhance data interoperability by enabling the integration of linked data into JSON based systems.

The vocabulary is published through a GitHub repository, which contains the relevant JSON-LD context files and corresponding examples of entities/assets. The context files define the mapping between short-form terms and their full IRIs, in accordance with the SEDIMARK ontology and the Smart Data Models initiative.

The repository is linked to a GitHub Pages service, which allows the JSON-LD contexts to be resolved as persistent HTTP URIs. These URIs can be referenced directly by assets, enabling automated interpretation of semantic annotation.

This publication mechanism ensures that:

- The structure and semantics of the vocabulary are explicitly defined and consistently applied.
- Contexts can be dereferenced dynamically as part of data exchange processes.
- Example entities illustrate correct usage patterns and support developer adoption.

2.5 Workflow Asset Transformer

The MageToCWLTransformer is a tool that converts Mage.ai pipelines into Common Workflow Language (CWL) workflows and standard Python scripts, facilitating the integration of user-friendly design environments with industry-standard execution frameworks. This Asset bridges

the gap between intuitive pipeline prototyping in Mage.ai and reproducible, portable deployment environments enabled by CWL [21].

The tool is composed of two main components:

- MageToPython which converts Mage.ai blocks into standalone Python scripts by removing Mage-specific dependencies, resolving environment variables, and ensuring direct command-line execution compatibility.
- MageToCWL that wraps the transformed Python blocks into CWL tools and workflows using structured templates. It builds CWL-compliant execution sequences and outputs YAML workflows, shell wrappers, and validation scripts.

The output is a ready-to-use ZIP archive containing:

- Converted Python scripts for each Mage block,
- CWL tool definitions and a main CWL workflow,
- A shell script for execution and validation

Optionally, serialization and visualization components (e.g., pickled data states, result displayer). The transformer supports seamless integration with CWL-WES and TESK [13] execution backends, use in cloud-native environments including Kubernetes and EDC connectors as well as enhanced reproducibility for Mage-authored workflows in federated or regulatory-constrained settings. As presented in Figure 18, we have performed tests, namely the average execution time over five consecutive runs for the two pipelines described previously.

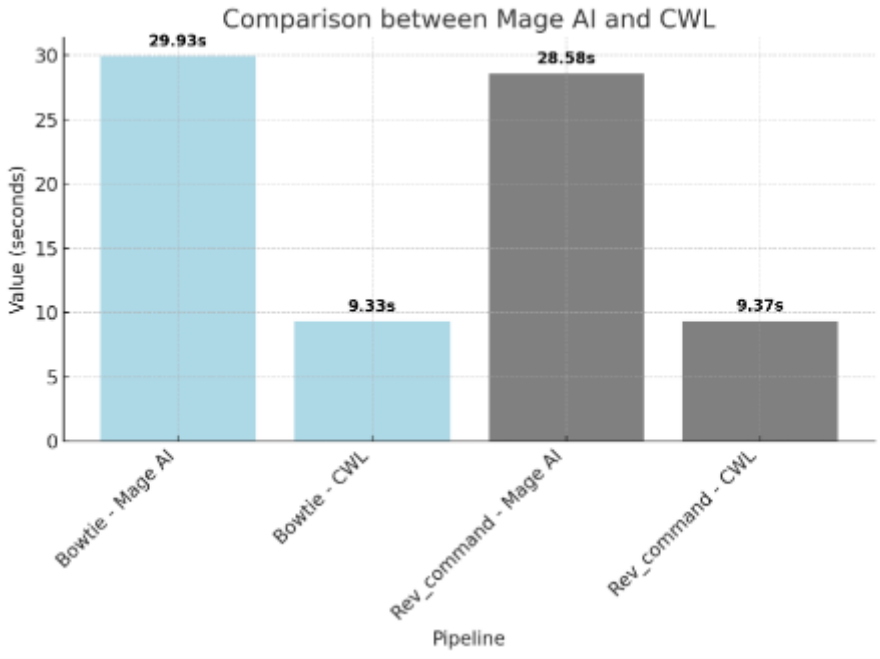


Figure 18: Comparison between Mage.ai and CWL performances

The results showed that CWL is faster in execution than Mage AI, which has a large overhead, presenting the benefits of using CWL for workflow execution.

This asset is aligned with the SEDIMARK objective of standardizing workflow interoperability across heterogeneous ecosystems.

It will be published as a WorkflowAsset under the type sedimark:PipelineConversionTool, linked to its Mage.ai source and CWL output via provenance metadata.

2.6 Offering Generator

The Offering Generator is a component that transforms asset information into standardized JSON-LD offerings compliant with the SEDIMARK Marketplace Information Model. It leverages Large Language Models (LLMs) to automate the creation of semantically rich offering structures.

In the SEDIMARK workflow, the Offering Generator receives asset details from providers and produces properly formatted offering documents that the Offering Manager can register in the marketplace. By using LLMs, this component significantly reduces the technical knowledge required from providers to create valid offerings while ensuring compliance with the complex ontological requirements described in section 2.3.

Recent advances in LLMs have demonstrated increasing proficiency in generating structured data with advanced techniques such as specialized schema usage, prompt engineering, and routing mechanisms. Leveraging LLMs for structured offering data generation enforces consistent semantics and structure, enhances interoperability across the marketplace, enables providers to describe offerings in natural language, drastically accelerates creation, reduces technical barriers, and minimizes the need for specialized knowledge or manual validation.

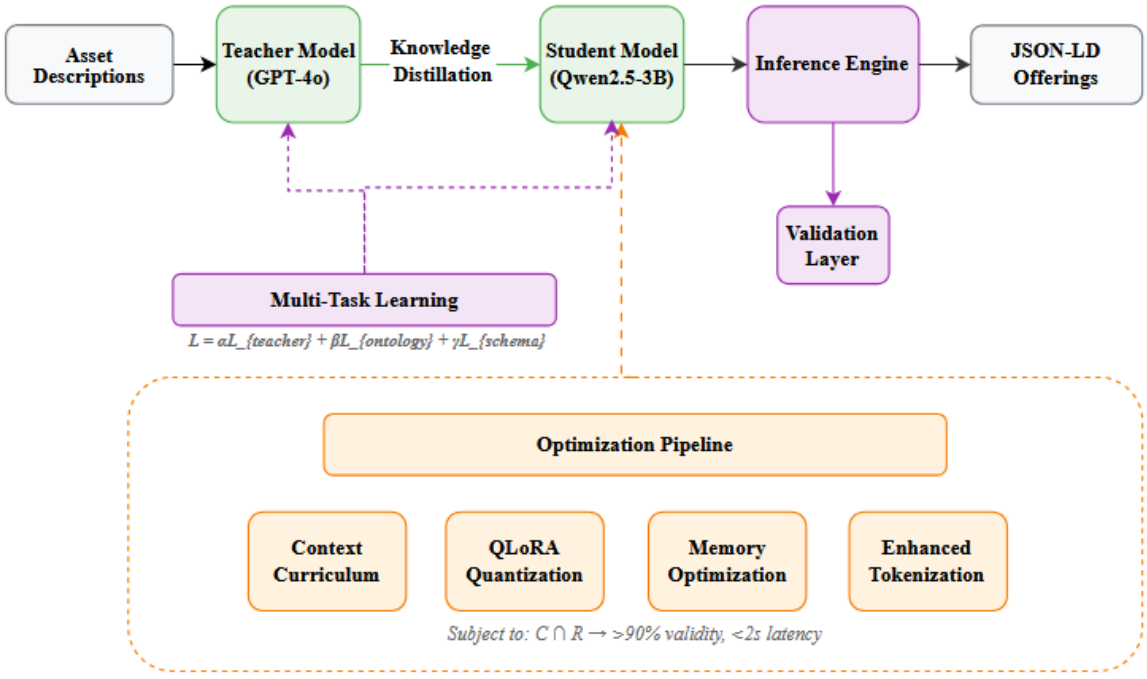


Figure 19: Handling Unstructured Metadata as Context for LLM

Functional description of component

The Offering Generator converts provider input and asset metadata into JSON-LD documents through prompt-engineered interactions with an LLM, ensuring alignment with the Marketplace Information Model. The generated output includes appropriate context declarations, semantic relationships, and contractual terms required for marketplace transactions. These documents

are subsequently validated against SEDIMARK schemas, with corrections applied as needed to ensure semantic consistency and interoperability within the marketplace ecosystem.

Interaction with Offering Manager

The Offering Generator interacts with the Offering Manager through a structured communication protocol to ensure seamless integration within the SEDIMARK ecosystem. The interface the Offering Manager exposes is a REST-based API that takes the generated Offering Description as the payload, and responds with a JSON payload that confirms the validity of the Offering Description, and it's storage in the Self-Listing. Else if the validation of the Offering Description fails, it will respond with a JSON payload that includes the error, whether it be basic RDF/JSON-LD compliance, or the Offering Description being incomplete. Following the JSON-LD generation and validation processes, the Offering Generator will forward the compliant offering documents to the Offering Manager through a dedicated API endpoint. The Offering Manager will receive these validated offerings and perform several critical functions, such as storage in the corresponding Self-Listing and registering the Offering with the DLT Registry. This interaction will follow a protocol where the Offering Generator will transmit both the offering content (JSON-LD document) and metadata about the validation results.

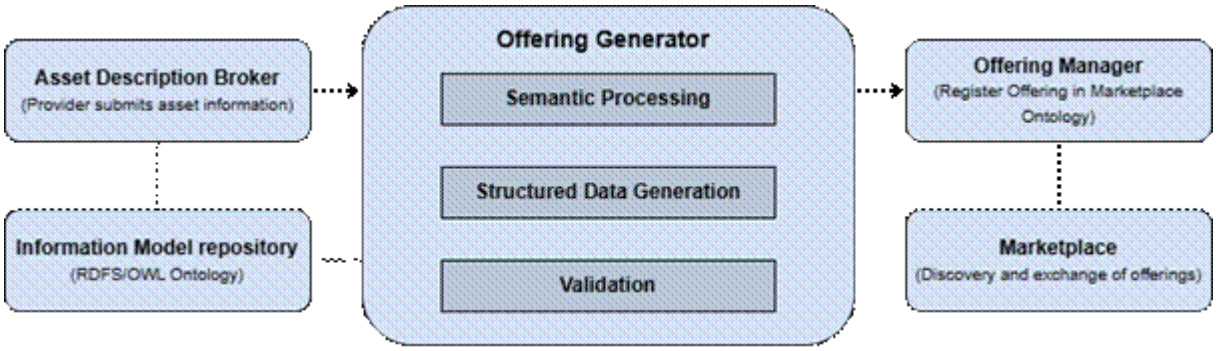


Figure 20: Interaction of Offering Generator with Offering Manager

Architecture of Offering Generator

At its core, this module employs a student-teacher architecture where large, computationally intensive models (like GPT) serve as teachers that produce structured JSON-LD examples, while a compact, efficient model (Qwen 2.5-3B) learns to replicate these capabilities through sophisticated knowledge distillation techniques. The distillation process employs curriculum learning that progressively challenges the student model, starting with complete contextual information and gradually reducing this support until the model can generate valid JSON-LD even with minimal context.

The operational pipeline begins with contextually-aware prompt engineering that provides carefully structured instructions to the base models. This critical first stage implements schema-aware prompting techniques that precisely define the expected JSON-LD structure, relationships, and ontological constraints. A high-capacity teacher model, e.g., a GPT variant, is leveraged to synthesize high-quality, schema-conformant JSON-LD instances. Each example is validated against schema requirements using evaluation metrics.

In the next phase, the student model is trained using a progressive context curriculum where training begins with complete ontologies and schemas, then gradually reduces contextual support through five distinct stages with increasing context dropout rates. The multi-stage

optimization process employs gradient accumulation, a cosine learning rate scheduler, adaptive learning rates, and mixed precision training. Advanced regularization techniques, including label smoothing, tiered dropouts, and stochastic depth, further enhance the model's performance.

The final phase focuses on refinement and specialization, emphasizing structural refinement and context retrieval training. During structural refinement, the model undergoes fine-tuning specifically on complex JSON-LD structures, focusing on proper entity linking, ontology compliance, and schema validation. Context retrieval training teaches the model to identify relevant contextual information through input-context-output examples and similarity metrics. The process also incorporates scheduled sampling to reduce exposure bias between training and inference.

The technical implementation includes several optimizations to ensure efficient training and deployment. Memory optimization techniques include gradient checkpointing and 4-bit quantization using NF4 format during inference. The model architecture benefits from QLoRA fine-tuning with specific parameters and differential learning rates for different layer groups.

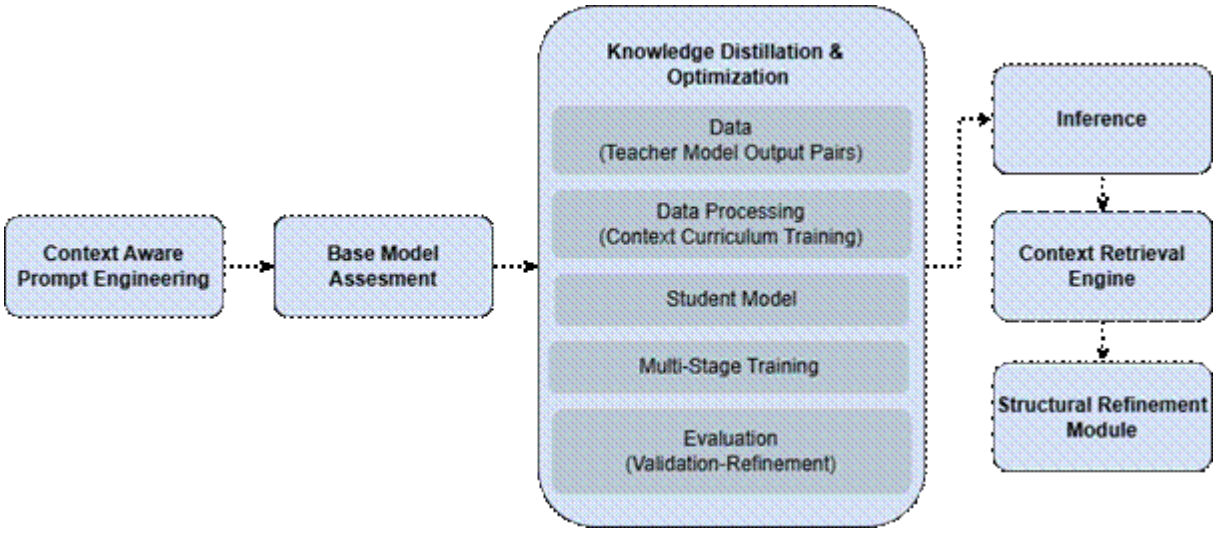


Figure 21: Offering Generator Pipeline

3 The Interoperability enabler

3.1 Data Formatter

Data Formatter is an essential component of the Interoperability Enabler, transforming data from various formats, such as CSV, XLS, XLSX, and NGSI-LD json, into the SEDIMARK internal processing format, specifically pandas DataFrames. This process is designed to standardize data input formats within the SEDIMARK ecosystem, ensuring that heterogeneous data can be seamlessly ingested and converted into a uniform structure. In this way, data formatting facilitates efficient data processing, analysis and integration, improving the overall functionality and reliability of the SEDIMARK system.

This component can automatically detect the file type based on its extension and uses the appropriate method to load the data. For CSV and Excel files, it uses pandas' built-in readers. When dealing with NGSI-LD json (primary standard format within the SEDIMARK ecosystem), which often contains deeply nested structures, the component applies a recursive flattening process to transform complex entities into flat, tabular records. It is designed to handle complex NGSI-LD json data more effectively than the *pandas.json_normalize* Python library. It recursively flattens dictionaries while preserving key hierarchies, supporting nested structures and ensuring efficient processing and interoperability.

This component enables comprehensive data management and seamless integration within the SEDIMARK ecosystem. The final output is a flat dictionary where complex nested structures are simplified, making it significantly easier to analyze and manipulate within a pandas DataFrame. This method enhances data accessibility and streamlines the analytical process.

Example with nested dictionary (e.g. single bike use case)

Input NGSI-LD json:


```
"id": "urn:ngsild:Vehicle:vehicle:MobilityManagement:196636",
"type": "Vehicle",
"category": {
    "type": "Property",
    "value": "tracked"
}
```


Figure 22: example NGSI-LD input for nested dictionary

Output (pandas DataFrame):

id	type	**category.type**	**category.value**
urn:ngsiLd:VehicLe:vehicLe:MobilityManagement:196636	Vehicle	Property	tracked

Figure 23: example DataFrame output of a NGSI-LD nested dictionary

Example with list handling in NGSI-LD json (e.g. temporal bike use case)

If a list contains dictionaries (list of dictionaries), each entry is flattened with an indexed key. The lists of primitive values are kept as separate indexed keys

Input NGSI-LD json:

Figure 24: example NGSI-LD input of a temporal value.

Output (pandas DataFrame):

battery[0].type	battery[0].value	battery[0].instanceId	battery[0].observedAt	battery[0].unitCode	...
Property	1	instanceid:b816b94d-cf8c-445a-bc17-b3e0dfbca8da	2024-09-25T04:30:06Z	P1	...

Figure 25: example DataFrame output of a NGSI-LD temporal value

Example by preserving specific keys like @coordinates (e.g. temporal station use case):

Input NGSI-LD json:

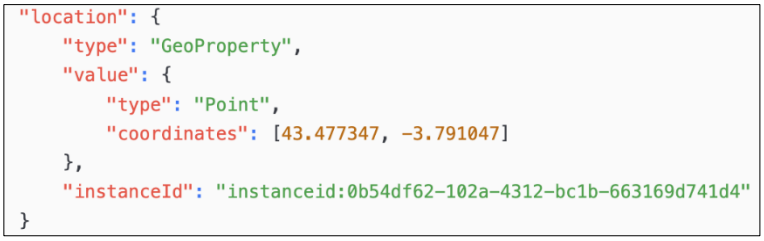


Figure 26: example NGSI-LD input of a geoproperty value.

Output (pandas DataFrame):

location.type	location.value.type	location.value.coordinates	location.instanceId
GeoProperty	Point	[43.477347, -3.791047]	instanceid:0b54df62-102a-4312-bc1b-663169d741d4

Figure 27: example DataFrame output of a NGSI-LD geoproperty

3.2 Data Curation

In the realm of data processing and analytics, the utilization of smart data models within the NGSI-LD data format has emerged as an approach for data annotation and enrichment. This section explores the dual facets of data annotations: global annotations, applied at the dataset level, and local annotations, which focus on individual data points.

Leveraging NGSI-LD's semantic capabilities and the richness of smart data models, this methodology ensures meaningful and interoperable annotations for improved comprehension and utilization of data.

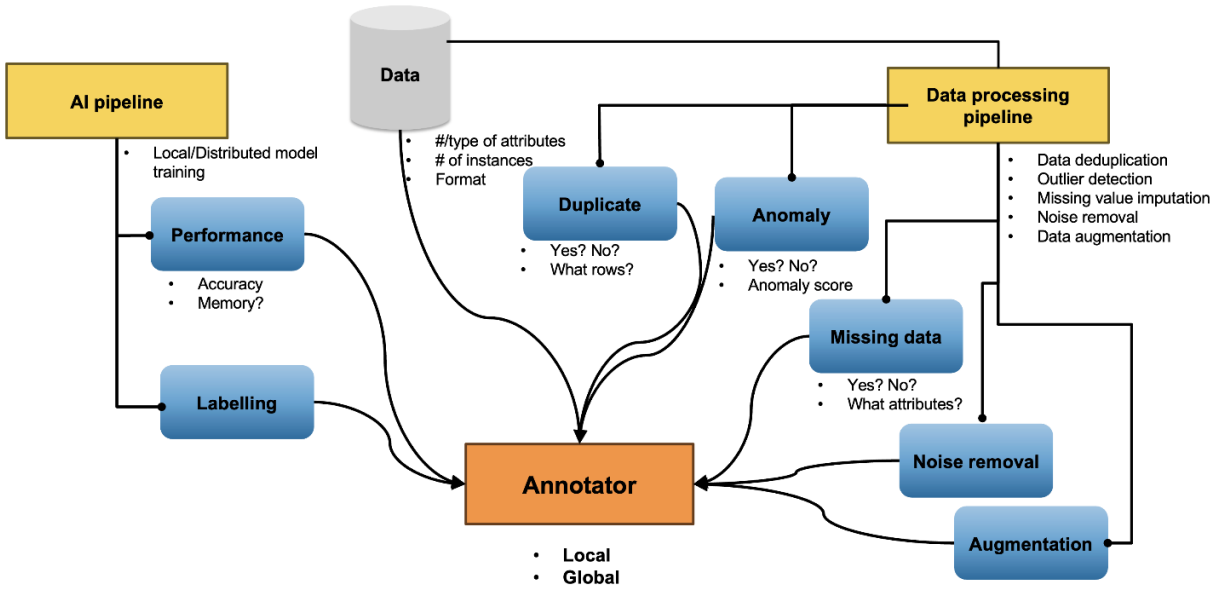


Figure 28: flow of local and global annotations

Annotations, whether at the global level, providing global information regarding datasets, or at the local level, enhancing individual data points with context-specific information, contribute to a more meaningful and interoperable SEDIMARK ecosystem.

3.2.1 Local annotations: enhancing individual data points metadata

Local annotations play a crucial role in enriching the metadata of individual data points with, *inter alia*, specific labels derived from data processing outcomes and specifically incorporating data quality models. This includes categorizing data points based on predefined criteria, enabling users to identify patterns, missing values, or anomalies. For instance, in SEDIMARK, we will use the anomaly scores and other data quality measures to support local annotations by adding metadata to mark data points that deviate significantly from the expected patterns. These annotations are crucial for identifying potential errors, anomalies, or noteworthy events that may require special attention. This information will be obtained from the Data processing and AI pipelines.

Local annotations also consider temporal aspects, capturing changes in individual data points over time. This temporal context enhances the understanding of the dataset dynamics, supporting applications that require historical analysis or real-time monitoring. In addition to standardized metadata, local annotations enable the inclusion of custom metadata tailored to specific SEDIMARK use cases. This flexibility allows users to embed domain-specific information, enhancing the richness of annotations for individual data points.

The integration of a data quality model to enrich the data within SEDIMARK refines this process by emphasizing the accuracy and completeness of individual data points and including information about outliers, missing data, and other anomalies. As shown in Figure 28, this metadata will be mainly generated from the Data deduplication, Outlier detection, and Missing value imputation components. For this to happen, the Smart data model "Data Quality" will be

used to enrich the content of data points by matching the output of the data processing and AI pipelines to the properties of the Data Quality model. The existing specific properties for the different quality aspects that will be considered within SEDIMARK are provided in D3.1 [14]. For example:

- Accuracy
- Completeness (considering the missing values: isMissing, whatAttribute)
- Outlier (isOutlier, outlier score)
- Duplication (isDuplicate, whatInstance)

3.2.2 Global annotations: enhancing datasets/data streams metadata

Global annotations involve enriching the metadata associated with an entire dataset or data stream, providing a holistic view of the underlying information. This process is important for establishing a contextual foundation that facilitates a comprehensive understanding and utilization of the data as a whole. Smart data models with their domain-specific ontologies offer a structured semantic context for datasets, encapsulating the essential characteristics of the data.

Global annotations contribute contextual information to the dataset, offering insights into the overall purpose, source, and relevance that illuminate the overall data quality. Metrics such as completeness, accuracy, precision, and timeliness are essential components of global annotations, enabling users to assess the reliability of the dataset as a whole. This metadata enrichment facilitates efficient data discovery, sharing, and utilization in applications and analytics. Global annotations encompass general properties related to datasets or data streams and are presented in Sections 3.5 and 3.6 in the deliverable D3.1. For instance, we cite:

- Accuracy
- Precision
- Completeness
- Statistics extracted from data (data format, number of attributes, number of instances)
- Information regarding the dataset usage (e.g., with which ML task this data can be used, isLabeled)
- If data is curated (information on how outliers are identified and handled, how missing values are handled)

In this context of global metadata, DCAT is used within SEDIMARK as an integral element of the Offering description. Its role is to augment information about the Offerings, providing descriptions of datasets and any pertinent information required for enhanced data discoverability. Consequently, global annotations will be integrated within the Offering description component.

3.3 Data Quality Annotations

The Data Quality Annotations is designed to enrich pandas DataFrames (from the previous phase: Data Curation) by adding quality annotations, which are essential for ensuring data integrity, reliability, and interoperability across different artefacts, including data, AI models, and service offerings.

Built to operate directly on Pandas DataFrames, this component supports two levels of annotation granularity:

- Entity-level, where a quality descriptor is attached to the entire data point (i.e., the row);
- Attribute-level, where specific columns (i.e., attributes) within each record are individually annotated.

It follows the NGSI-LD standard for linked data, guaranteeing compatibility with decentralized and federated data architectures. It checks for and generates key metadata fields (*id*, *type*, and *@context*) if they are missing. It introduces the property "*hasQuality*" with the type "Relationship" and the object which specifies a unique identifier (URN) with the data entity (either an instance or an attribute) to uniquely identify entities and their associated quality assessments.

The URN follows this specific pattern:

- For attribute-level annotations:

  urn:ngsi-ld:DataQualityAssessment:<annotation_name>:<instance_id>:<attribut_name>

- For entity-level annotations:

  urn:ngsi-ld:DataQualityAssessment:<annotation_name>:<entity_id>

Examples with the attribute-level annotations (e.g. temporal station use case)

In this example, the selected attribute for annotation is "availableBikeNumber", and the corresponding annotation entity type (representing the metadata type of the instance) is "BikeDockingStatus". Since an NGSI-LD json file can contain multiple instances of different types, specifying the entity type is essential to accurately associate quality annotations with the correct data entity.

availableBikeNumber[0].hasQuality.type	availableBikeNumber[0].hasQuality.object
Relationship	urn:ngsi-ld:DataQualityAssessment:BikeDockingStatus:urn:ngsild:BikeHireDockingStation:bikestation:MobilityManagement:336926289:availableBikeNumber

...

availableBikeNumber[99].hasQuality.type	availableBikeNumber[99].hasQuality.object
Relationship	urn:ngsi-ld:DataQualityAssessment:BikeDockingStatus:urn:ngsild:BikeHireDockingStation:bikestation:MobilityManagement:336926289:availableBikeNumber

We have also the possibility to annotate a specific attribute at a granular-level.The attribute (metadata) chosen is "availableBikeNumber[0]".

availableBikeNumber[0].hasQuality.type	availableBikeNumber[0].hasQuality.object
Relationship	urn:ngsi-ld:DataQualityAssessment:BikeDockingStatus:urn:ngsild:BikeHireDockingStation:bikestation:MobilityManagement:336926289:availableBikeNumber

Example with the entity-level annotations (e.g. single bike use case)

In this example, only the entity type "FleetVehicleStatus" is selected for annotation.

...	hasQuality.type	hasQuality.object
...	Relationship	urn:ngsi-ld:DataQualityAssessment:FleetVehicleStatus:urn:ngsild:Vehicle:vehicle:MobilityManagement:196636

3.4 Data Mapper

The Data Mapper component is designed to convert the enriched pandas DataFrame (from the previous phase: Data Quality Annotations) back to NGSI-LD json, enabling seamless integration with the NGSI-LD Broker. During this transformation, it restores the original NGSI-LD structure, including nested attributes and contextual metadata, ensuring consistency with the source format.

To support incomplete or flat data sources, this component also generates missing semantic elements. If the *id* is absent, a default URN-based identifier is created following the pattern:

urn:ngsi-ld:{*entity_type*}:{DataFrame_row_index}.

If the entity type is not provided, it defaults to the specified entity_type parameter.

When attributes contain "type": "null", they are automatically corrected to "type": "Property" to conform to NGSI-LD standards.

Additionally, the component handles timestamp normalization, converting raw Unix timestamps or numeric date fields into the standardized ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ), such as 2024-04-17T00:00:00Z, to ensure temporal consistency across interoperable systems.

3.5 Data Extractor

The Data extractor component is to extract and return specific columns from a pandas DataFrame (from the Data Formatter component) based on the indices provided by the user. This functionality is crucial for enabling selective data processing and enhancing interoperability among various artefacts, including datasets, AI models, and service offerings. By delivering both the filtered DataFrame and the corresponding column names, the component ensures that downstream components (for examples Data Mapper, Data Quality Annotators, or AI service) can operate with the exact data they need. This capability supports the modular, flexible, and traceable data workflows that are essential for seamless integration within the SEDIMARK ecosystem.

3.6 Metadata Restorer

The Metadata Restorer component is designed to restore essential metadata, particularly column names, into a pandas DataFrame containing prediction results from an AI model.

In many AI processing pipelines, especially those involving raw numerical arrays or anonymized data, original column headers are often removed for performance or compatibility reasons. This component addresses the crucial need to restore this contextual information once predictions are complete.

By aligning a provided list of column names with the structure of the prediction output, it ensures that the resulting DataFrame is both human-readable and machine-interpretable.

This restoration process supports traceability and consistency within the broader SEDIMARK ecosystem. The component includes verification mechanisms to ensure that the number of columns in the prediction results matches the provided metadata, guaranteeing robust and reliable reintegration of information.

3.7 Data Merger

The Data Merger component is responsible for combining the original input data with the corresponding prediction results from an AI model. It ensures that the two DataFrames, the initial DataFrame (from the Data Formatter) and the DataFrame containing the predicted data with restored column names (from Metadata Restorer component), are aligned by their column names.

To facilitate a seamless merge, the function first identifies the union of all column names present in both DataFrames. For any columns missing in either DataFrame, it adds these columns and fills them with the string "NaN" to clearly indicate missing data.

The component then aligns both DataFrames to a consistent column order, sorted alphabetically, and concatenates them by row into a single unified DataFrame. This is particularly valuable in workflows where prediction results need to be reintegrated with the original dataset for further analysis, visualization, or exporting.

3.8 Data validation / certification

Validation is required to ensure that the formatting applied to data assets and Marketplace self-descriptions is valid and complies with their respective information models.

For both types of artefacts, validation is done through a set of stages.

- Format: the format that is used for representation complies with an acceptable serialization format and variant within that format.
- Syntax: the syntax applied to the annotation of the artefact complies with the classes and properties defined in the corresponding information model.
- Semantic: the axioms defined in relation to the relationships between instantiations of the concepts defined in the corresponding information model are compliant. This would include relationships regarding properties, class hierarchies, cardinalities etc.
- Domain-specific: the literal values that represent qualitative and quantifiable properties are valid in terms of ranges and states.

Table 3: artefacts and their validation process

Artefact	Information model	Format	Artefact	Information model	Format
Self-Description	Marketplace Information model	JSON-LD/RDF schema validator	RDF model validator	Ontology compliance checker	Not applicable
Data Asset	NGSI-LD, Smart Data Models	JSON-LD schema validator	JSON-schema based validator, NGSI-LD model validator	SHACL validator (for graph-based validation)	Domain ontology + taxonomy validator

3.8.1 Offering Description Validation

When Offerings are submitted to the Offering Manager, the Offering Manager will check the validity of the Offering Description with the Validation Suite. The document will be checked syntactically in that it complies with the JSON-LD schema and is compatible with the RDF schema. The next step is that it checks that it complies with concepts defined in the SEDIMARK ontology. It will then check that the minimum required instances of classes and their properties are provided. To enable this, SHACL validation will be used for this purpose. The document first should be viewed from an Offering centric perspective, meaning that any validation starts with the Offering Class. The mandatory requirement for an Offering document must have the following:

Table 4: SHACL Shape Rules for Offering Validation

Class	Property	Value	Shape (Rule)
Offering	dcterms:title	xsd:string	must have one
	dcterms:description	xsd:string	must have one
	dcat:themeTaxonomy	skos:ConceptScheme	must have at least one, and must exist in document
	sedimark:hasAsset	sedimark:Asset	must have at least one, and must exist in document
	sedimark:isListedBy	sedimark:Self-Listing	must have at least one, and must exist in document
	sedimark:hasOfferingContract	sedimark:OfferingContract	must have at least one, and must exist in document
	dcterms:license	xsd:string	must have one

Class	Property	Value	Shape (Rule)
Asset	sedimark:offeredBy	sedimark:Offering	must have at least one, and must exist in document
	sedimark:isProvidedby	sedimark:AssetProvision	must have at least one, and must exist in document
	sedimark:hasAssetQuality	sedimark:AssetQuality	must have at least one, and must exist in document
	dcterms:theme	skos:Concept	must have at least one, and must exist in document
	dcterms:identifier	xsd:string	must have one
	dcterms:title	xsd:string	must have one
	dcterms:description	xsd:string	must have one
	dcterms:creator	xsd:string	must have one
	dcterms:issued	xsd:string	must have one
	dcat:keyword	xsd:string	must have at least one
	dcterms:spatial	xsd:string	only one
	prov:generatedBy	xsd:dateTime	can have one
	dcat:isVersionOf	xsd:dateTime	can have one
Self-Listing	sedimark:belongsTo	sedimark:Participant	must have only one, and must exist in document
	dcterms:title	xsd:string	must have one
	dcterms:description	xsd:string	must have one
	dcterms:issued	xsd:dateTime	must have one
	dcterms:modified	xsd:dateTime	must have one
	schema:accountId	xsd:string	must have one
	schema:email	rdf:Resource	must have one
OfferingContract	odrl:permission	odrl:Duty	must have one
	odrl:duty	odrl:Duty	must have one
	odrl:obligation	odrl:Duty	must have one

The equivalent SHACL description is provided in Annex 9.1.

4 The AI enabler

The AI Enabler of SEDIMARK is a core component designed to empower decentralized, privacy-preserving, and energy-efficient AI workflows. It integrates tools and services to allow participants to train, deploy, and optimize machine learning models locally or collaboratively through federated learning, without sharing raw data. The AI Enabler is delivered as a modular, containerized toolbox, fully deployable via Docker, Docker compose, and optionally Kubernetes for scalable environments. The modularity ensures that each organisation or participant can selectively deploy the services they need, according to their infrastructure, use case, and privacy requirements.

From a user perspective, the AI Enabler allows to:

- Train models locally on private datasets
- Join a federated learning training session without moving data
- Optimise models for energy efficiency and transferability across environments
- Deploy services via simple commands using Docker/Compose or Kubernetes manifests

4.1 Interoperable Federated Learning for SEDIMARK

4.1.1 Introduction

Federated Learning (FL) is a key paradigm shift introduced in SEDIMARK to enable AI collaboration without centralized datasets. Instead of uploading sensitive data to a central server, participants train models locally and only share model updates. These updates are then securely aggregated to build a shared global model. In SEDIMARK, we focus on interoperable FL, meaning that:

- Different organisations with different platforms and infrastructures can still collaborate
- Model updates follow standard formats like ONNX, TensorFlow to ensure compatibility
- Communication between nodes is handled through secured APIs.

As a user aiming to participate in a FL session within the SEDIMARK framework, the process begins by deploying the local FL agent using the provided Docker Compose file or Kubernetes manifest. Once deployed, the user configures the client to point to their local dataset folder and defines key training parameters, such as batch size and learning rate, typically through an *.env* file or a YAML configuration. The client is then connected to a Federated Learning server, also provided as part of the SEDIMARK Toolbox, by specifying the appropriate API endpoint. After setup, the client will execute training locally on the user's infrastructure and securely share encrypted model updates with the FL server. This process enables users to collaboratively build high-quality AI models while fully preserving data sovereignty, ensuring that raw data remains private and on-premise.

4.2 Local model training

Local model training remains essential when data must not leave the organization's premises or when specific edge applications are targeted. This service is part of the AI Enabler, allowing organizations to independently train and optimize AI models while preparing them for potential future federated learning participation.

In the SEDIMARK marketplace, users are empowered to deploy tailored AI model training pipelines suited to specific use cases—such as energy consumption forecasting or customer churn prediction. These pipelines are designed for scenarios where users have access to local datasets, enabling model training to occur entirely on-premise or within a trusted environment. This approach ensures data privacy and regulatory compliance by eliminating the need to share raw data externally.

Each AI pipeline is accessible through the SEDIMARK Marketplace interface as a selectable service tailored to a specific use case, such as energy consumption prediction or customer churn analysis. Once selected, the pipeline is deployed as a containerized microservice within the user's trusted execution environment. The pipeline follows a predefined sequence of steps including secure data loading, preprocessing using certified processors (e.g., normalization, feature encoding), local model training with configurable algorithms, and model evaluation based on relevant performance metrics. This setup ensures full control over data privacy while enabling reproducible, high-quality machine learning workflows.

To better illustrate this process, the figure below presents the structure and operational flow of two AI pipelines provided via the SEDIMARK marketplace: one for energy consumption prediction and another for customer churn analysis. Both pipelines are designed to execute entirely within the user's local environment, using their own datasets and resources inside the SEDIMARK's Marketplace User Interface (UI).

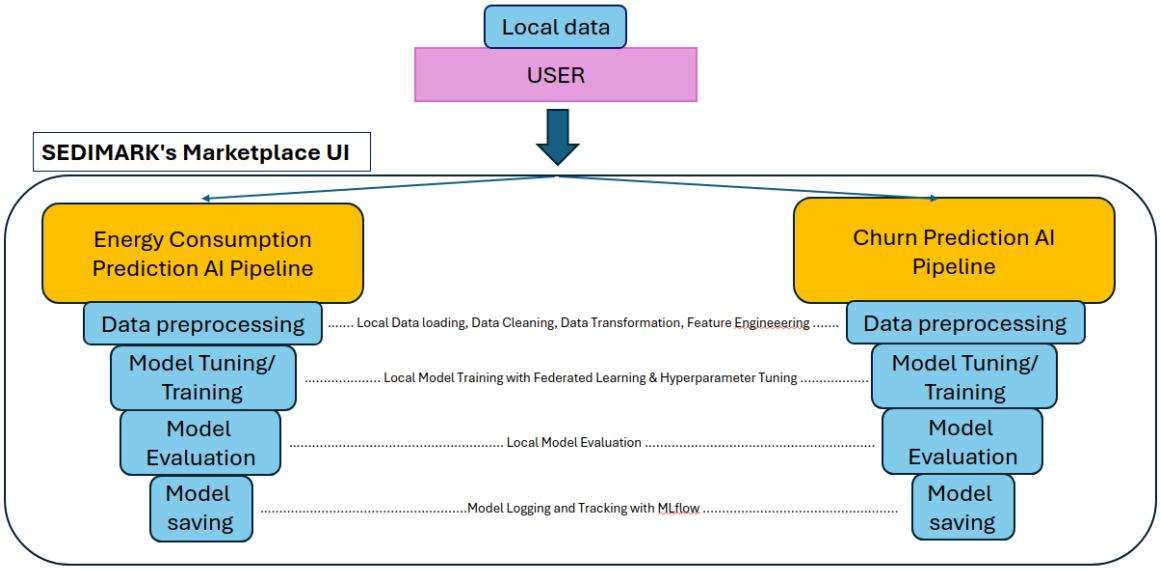


Figure 29: Structure and operational flow of two AI pipelines provided in the marketplace

Each pipeline Is composed of four main stages:

- **Data processing**: The user's local data is securely loaded and passed through certified preprocessing components, including steps such as data cleaning, normalization, transformation, and feature engineering. These steps ensure consistency and readiness of the dataset for downstream machine learning tasks.
- **Model tuning and training**: The pipeline performs local model training using configurable machine learning algorithms, optionally enhanced with automated hyperparameter tuning. In scenarios involving Federated Learning, the training process is extended to a collaborative setup where the user's environment acts as a decentralized node. Each

node trains the model on its local data and periodically exchanges model updates (not raw data) with an aggregator through secure protocols. This allows multiple users to contribute to a shared global model while preserving full data privacy and ensuring regulatory compliance. The federated approach leverages distributed knowledge across data owners without requiring central data storage.

- **Model evaluation**: The trained model is validated using performance metrics such as accuracy, F1-score, or RMSE, depending on the use case. This evaluation is done locally and can be used to compare multiple training runs.
- **Model saving and tracking**: Finally, the trained model and its associated metadata (e.g., hyperparameters, metrics, training duration) are logged using MLflow, providing a versioned, reproducible record of the experiment. This enables the user to track multiple runs, select the best model, and export it for further use or deployment.

The AI SEDIMARK pipeline will enable the building and training of an AI model.This is illustrated here with a model defined for energy consumption prediction.

In the electricity consumption prediction endeavour, we harness a week's worth of time-series energy consumption data, preceding our decision-making juncture, to forecast subsequent daily consumption in hourly intervals (Figure 30). Utilizing the advanced DeepAR model [15], we aim to construct a universal framework capable of accurately predicting consumption patterns across facilities and buildings of diverse magnitudes (Figure 37).

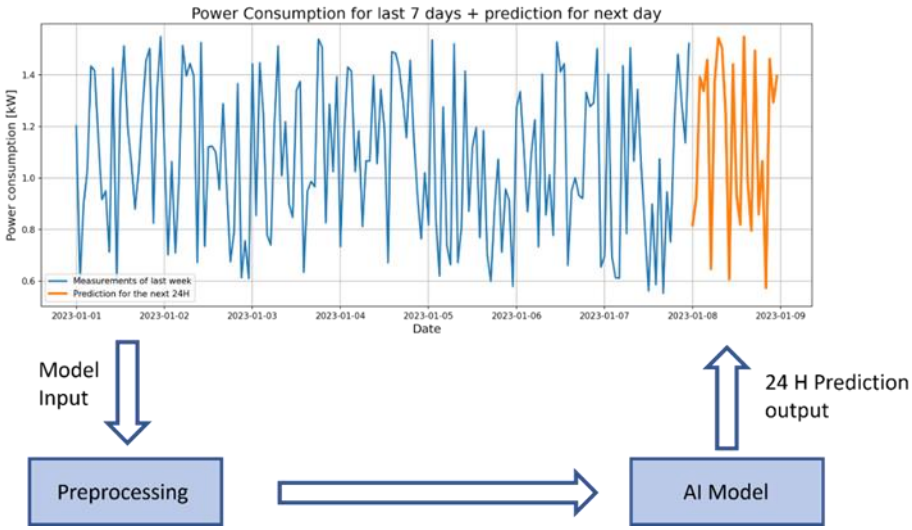


Figure 30: general principals of the locally trained predictive module

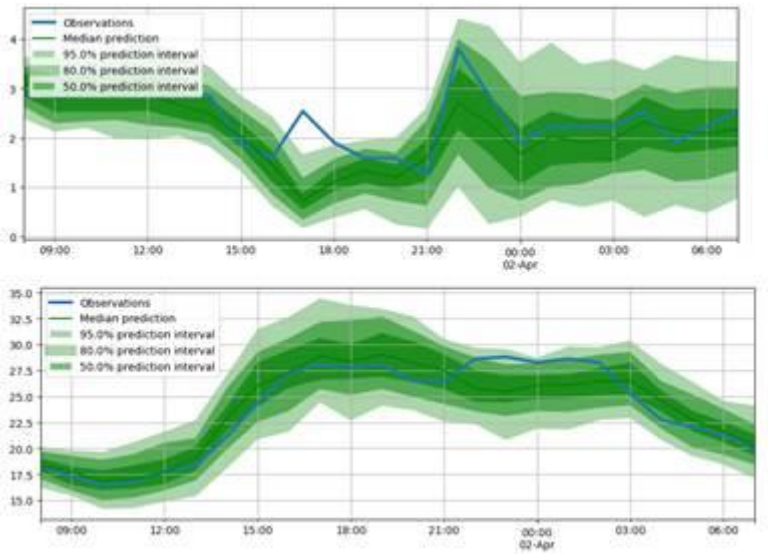


Figure 31: comparing deepAR based predictions with observations

Another example of a locally trained model relates to customer segmentation and churn prediction. In this initiative, we meticulously preprocess and sanitize datasets encompassing electricity consumption patterns, payment histories, geographical metrics, and behavioural indicators like complaints. Employing state-of-the-art ensemble decision tree algorithms such as LightGBM [16], Catboost [17], or XGBoost [18], our objective is to segment our customer base and forecast churn propensity, culminating in a calculated churn probability for each individual customer (Figure 32).

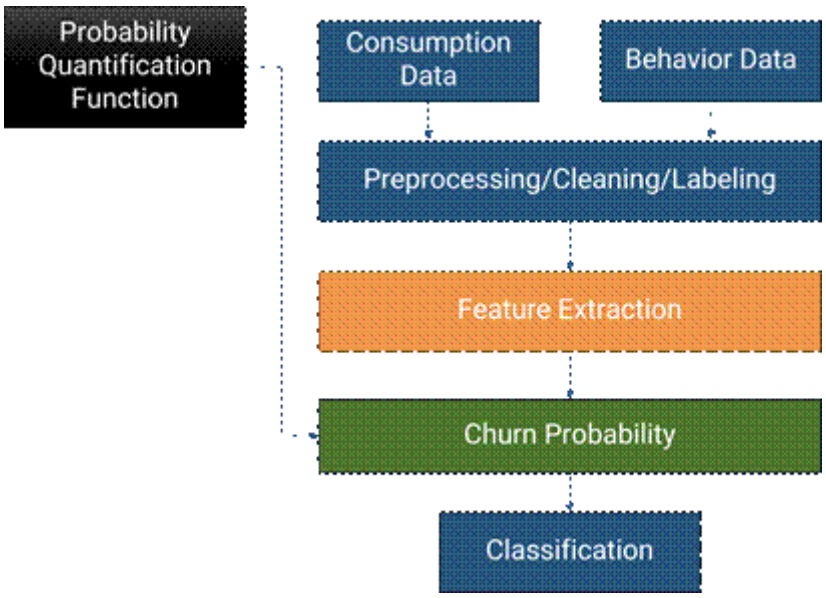


Figure 32: customer segmentation and churn prediction.

4.2.1 Times-series Multivariate Forecasting based on CrossFormer technique

SEDIMARK also provides the support for multivariate forecasting using the latest transformer based deep learning. Different to classical machine learning, CrossFormer [19] is built to forecast time-series multivariate by exploiting temporal and cross-variable (inter-dimensional) information. To achieve this, CrossFormer contains three key components:

- Dimension-Segment-Wise (DSW) Embedding: This method segments and embeds the input time series across dimensions, allowing the model to capture local temporal patterns more efficiently.
- Two-Stage Attention (TSA): The attention mechanism operates in two stages — first across time (to learn temporal dependencies within each variable), and then across dimensions (to capture relationships between variables).
- Hierarchical Encoder-Decoder (HED): A scalable architecture that helps manage long-range dependencies while keeping computation efficient.

With above components involved, CrossFormer is well suited for forecasting tasks in environments like SEDIMARK, where data is high dimensional, noisy and time-dependent. Furthermore, this technique is available as ready to use as Python package for further development and usage (not limited to SEDIMARK). Besides, this method can fit to diverse use cases with different configurations.

CrossFormer Component

As a component of the AI toolbox within SEDIMARK, the CrossFormer module is structured into two main parts, referring to Figure 20:

- Core Package: This includes the implementation of the CrossFormer model itself and the associated data interface. It handles model architecture, data preprocessing, and interaction with training/inference routines.
- Wrapper Scripts: These scripts provide high-level interfaces for training and inference. They are designed to be easily configurable and support automated integration into different use cases.

Together, these components enable CrossFormer to serve a variety of forecasting tasks within SEDIMARK. The modular design allows it to be reused or adapted across domains with different data formats or prediction requirements, supporting scalable and flexible AI-driven services.

Core Package includes main features of the algorithm, including model, evaluation, and data processing.

First, the model feature is implemented by PyTorch Lightning providing the model definitions, forward logistics and engineering interface support (training and inference). The model can be initialized with diverse configurations to fit different use cases.

Second, the evaluation feature compresses loss function and evaluation metrics, which is provided to evaluate the performance of model during training and validation. It includes MAE, MSE, RMSE, MAPE. MSPE. RSE, CORR, scaled MSE, normalized MSE, scaled Log Cosh and Hybrid Loss. The details of each function can be found in Annexes - Evaluation Metrics for CrossFormer. Based on those functions, we define the hybrid loss (score) with a controllable term for optimizing the model, which can be used to handle input values in diverse range. It is defined as $SCORE = \alpha ScaledMSE + (1-\alpha)ScaledLogCosh$. The purpose of the

Hybrid Loss is to balance forecasting on trending and exact estimates on each time step. Therefore, model can perform higher forecasting accuracy when applied with Hybrid Loss rather than simple MSE.

Third, the data feature provides the interface to handle general 2D data from different use cases with various data shape. It supports automatic data loading, batching and dataset setup.

The Wrap Script is another important component. It provides a convenient, standardized interface for using the CrossFormer model within data processing or pipeline environments such as SEDIMARK. It contains utility functions for model training (fit/setup) and inference, reducing boilerplate and abstracting away low-level details. As well, it enables the connection with MLFlow to register and load models inside Mage AI.

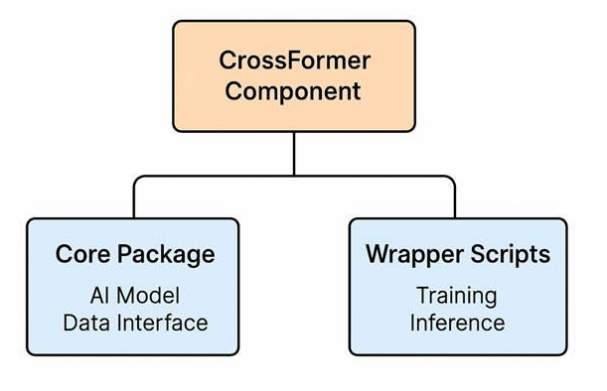


Figure 33: CrossFormer Component Overview

Summary of the CrossFormer Component

CrossFormer plays a key role in the AI pipeline of SEDIMARK, serving as the core forecasting module based on historical time series data. It processes 2D value-only data frames received from upstream blocks and supports both training and inference workflows. The model's behavior is governed by configuration files, allowing flexible adaptation to a wide range of use cases.

To support robust deployment and lifecycle management, MLflow [20] is integrated for model versioning, monitoring, and experiment tracking. During training, metrics and artifacts are automatically logged, and the final model is registered for subsequent inference tasks.

The Figure 34 below illustrates how the CrossFormer component fits into the training and inference flow within the AI toolbox.

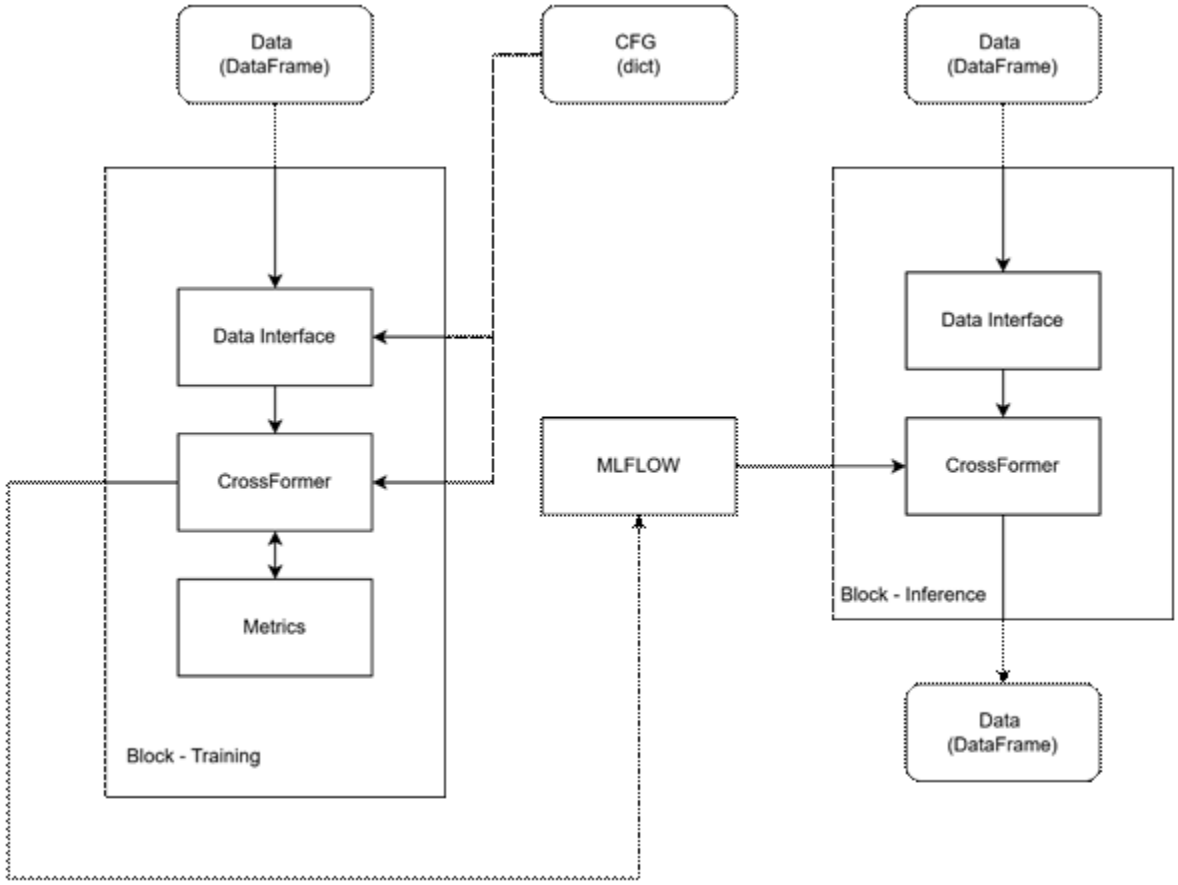


Figure 34: CrossFormer Component Workflow Overview

Crossformer Pruning for Efficient Time Series Forecasting

Crossformer Pruning is designed to reduce the computational complexity by removing redundant parameters without compromising performance. It plays a crucial role in optimizing model deployment for resource-constrained and federated environments, aligned with the goals of the SEDIMARK project.

SEDIMARK focuses on secure and efficient machine learning across distributed data ecosystems. The pruning module contributes by:

- Reducing model size for faster edge inference
- Lowering memory and energy consumption
- Enabling deployability in heterogeneous, low-power nodes

Pruning is a technique that removes less important parts of a neural network (such as weights or neurons), reducing the overall size and computational cost of the model. This process leads to more efficient models that can run faster and require fewer resources, often with little or no drop in predictive performance.

The following two pruning techniques are applied to reduce model complexity and improve efficiency:

- Unstructured Pruning: unstructured pruning removes individual weights from the network based on their importance. This type of pruning leads to sparse weight matrices and can significantly reduce parameter count.
  - Typically applied based on magnitude or statistical criteria.
  - Maintains model architecture but zeroes out less significant weights.
  - Requires sparse-aware hardware or libraries to realize speed or efficiency gains in practice.
- Structured (Channel) Pruning: Structured pruning removes entire neurons, channels, or attention heads, resulting in a physically smaller model with fewer operations (FLOPs). This is especially beneficial for hardware acceleration.
  - Applied at a higher architectural level.
  - Results in reduced model size and faster inference.
  - Maintains dense weight matrices, making it highly compatible with standard hardware like CPUs and GPUs

Combining unstructured and structured pruning provides a balanced trade-off:

- Unstructured pruning reduces redundancy in weights.
- Structured pruning optimizes the model for deployment.
- The result is a smaller, faster model that can still match original performance when fine-tuned.

The pruning module fits into Crossformer pipeline at the model optimization stage, after model loading and before training or inference.

- Input: Config file (JSON), training/evaluation data (CSV/DataFrame)
- Output: Pruned, fine-tuned, and MLflow-registered model

As illustrated in Figure 35, the pruning workflow is structured into stages distinguished by colour. The blue stages represent the initial setup, including loading the configuration and data, and initializing the data interface. The orange stages correspond to key model optimization operations, such as applying pruning (unstructured or channel pruning), saving the intermediate pruned model, reloading it, and profiling it for FLOPS and size. The purple stages cover the final training of the pruned model, registration with MLflow, and the testing or inference phase. This emphasizes the modular and iterative nature of the pruning process within the Crossformer pipeline.

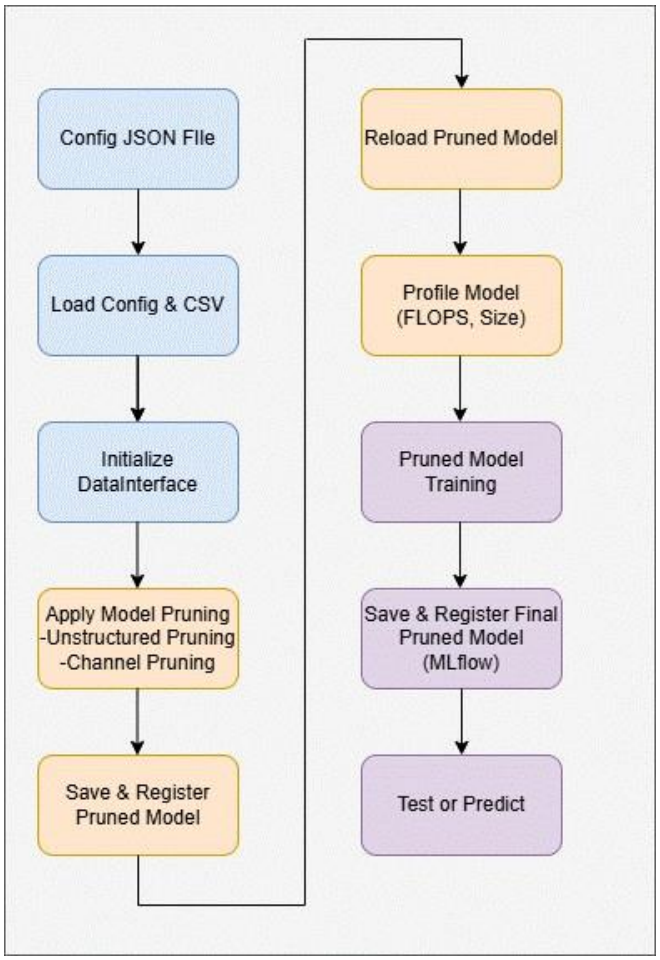


Figure 35: Crossformer Pruning Flow

Evaluations for KPIs, benchmarking scenarios, and comparison are documented in the Performance Evaluation Section of SEDIMARK_D3.2 [14].

4.2.2 Offering Generation training

The Offering Generator transforms asset metadata into semantically rich JSON-LD offerings aligned with the SEDIMARK Marketplace Information Model. By leveraging Large Language Models (LLMs) and sophisticated prompt engineering techniques, it bridges the semantic gap between natural language descriptions and structured marketplace offerings. This section details the technical implementation, prompt engineering methodology, and training approach used to develop this module.

Prompt Engineering Cycle

The prompt engineering cycle forms the foundation of the Offering Generation system and was established as the initial phase of development due to the inherent complexity of JSON-LD structure and the need for semantic precision in marketplace offerings. This approach was necessitated by the observation that even advanced LLMs struggle with producing consistently valid JSON-LD structures without proper guidance, particularly when dealing with complex relationship patterns required by the SEDIMARK ontology.

The system implements a cyclical five-stage prompt engineering process as illustrated in Figure 37

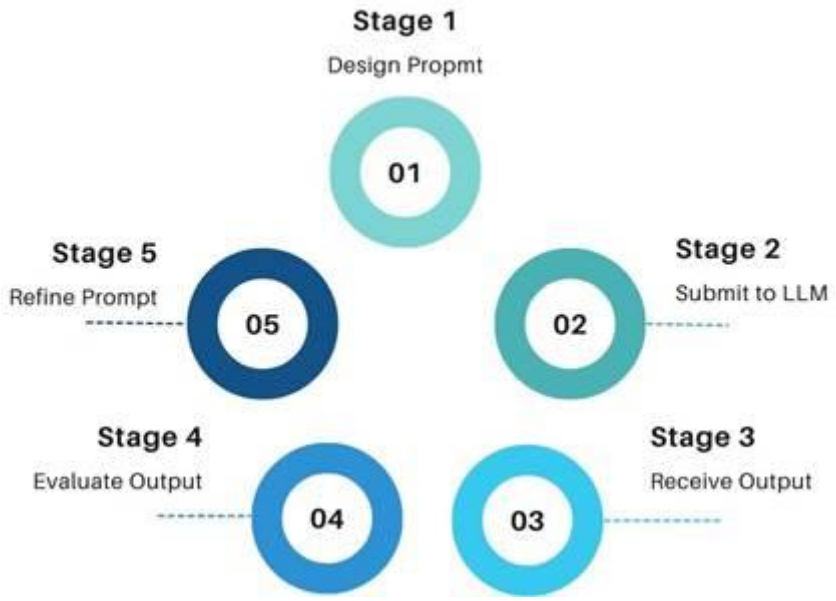


Figure 36: Five stages of Prompt Engineering process

Design Prompt: Engineer JSON-LD schema-aware prompts with explicit structural requirements, incorporating ontological constraints and relationship patterns from the SEDIMARK information model.

Submit to LLM: Present prompts to the model with appropriate context, using controlled temperature settings (0.7) to balance creativity with precision.

Receive Output: Capture generated JSON-LD structures and parse them for structural and semantic analysis.

Evaluate Output: Apply comprehensive evaluation metrics to identify pattern failures and semantic inconsistencies

- Exact Match Comparison is percentage matching of each field.
- Structural Similarity Index (SSI) measures structural resemblance.
- Custom JSON Diff identifies key-value differences.
- Semantic match validation assesses relationship correctness.

BLEU score measures n-gram overlap between generated structures and ground truth references, calculated as: $BLEU = BP \cdot e^{\sum_{n=1}^{N} w_n \log p_n}$ where BP is brevity penalty, w_n are weights, and p_n is n-gram precision

Refine Prompt: Iteratively improve prompts based on evaluation results, incorporating successful patterns and adding guardrails against common errors.

This cyclical approach was essential for developing a corpus of reliable prompt templates that could reliably produce valid JSON-LD structures conforming to the marketplace information model.

Zero-Shot Testing and Evaluation: After establishing baseline prompt patterns, comprehensive zero-shot testing was conducted to evaluate the generalizability of the approach and identify systematic failure modes. This phase was critical for understanding the

inherent limitations of pretrained models when dealing with structured JSON-LD generation tasks without domain-specific fine-tuning.

The system's evaluation framework measured performance across multiple dimensions as shown in section 2.6. Structural accuracy was assessed using automated validation against JSON-LD schemas, with successful generations typically achieving above 95% compliance. Semantic validity was evaluated through graph-based analysis of entity relationships, ensuring that connections like "hasAsset," "providedBy," and "references" were correctly established according to the marketplace ontology.

Initial testing across multiple LLM architectures (Claude, Gemini, Llama, GPT, Mistral) identified five key differences in zero-shot performance:

- Resource identifier pattern differences: Inconsistent URI formatting and namespace usage.
- Dublin Core Terms (DCT) implementation differences: Variations in property application and value representation.
- Entity hierarchy and relationship differences: Incorrect parent-child relationships and missing mandatory connections.
- Temporal representation differences: Inconsistent datetime formats and temporal relationship modelling.
- Structural component differences: Variations in entity organization and attribute placement.

Model Architecture and Pipeline

Based on the limitations identified in zero-shot testing, the system architecture was designed as a student-teacher framework where large, computationally intensive models serve as teachers that produce structured JSON-LD examples, while a compact, efficient model (Qwen 2.5-3B) learns to replicate these capabilities through knowledge distillation techniques.

The operational pipeline begins with contextually aware prompt engineering that provides carefully structured instructions to define the expected JSON-LD structure, relationships, and ontological constraints, following the marketplace information model. The multi-stage optimization process employs several advanced techniques:

Gradient accumulation: $g_t = \frac{1}{n}\sum_i^n g_t^i$ where g_t is the gradient from the i-th microbatch.

Label smoothing: $y_{LS} = (1-\alpha)\, y_{one-hot} + \frac{\alpha}{K}$ where α is the smoothing factor and K is the number of classes.

Tiered dropouts: Progressive dropout rates applied to different layers of the network, with higher rates for later layers.

Cosine learning rate schedule: $\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$.

The model was initialized with pretrained weights and then fine-tuned on a diverse corpus of JSON-LD examples gathered from the prompt engineering cycle. This approach significantly improved the model's ability to generate semantically valid and structurally correct offerings across diverse domains.

Context Dropout Strategy

A key innovation in the training methodology was the implementation of curriculum learning through a progressive context dropout strategy. This approach was necessary to ensure the model could generalize beyond the examples provided during training and handle novel input descriptions with limited or no contextual examples.

The context dropout rate was controlled by the following formula:

$$CD(t) = \min\left(C_{max}, C_0 \cdot \left(1 - \alpha \cdot {}^{t}/_{T}\right)\right)$$

Where,

- CD(t) is the context dropout rate at training step t
- C_{max} is the maximum dropout rate
- C_0 is the initial dropout rate
- α is the dropout increase factor
- T is the total number of training steps

This approach progressively challenges the student model with increasingly difficult context conditions:

- Initial phase: Full context availability (70% of training) to establish core pattern recognition
- Middle phase: Partial context with progressive dropout (20% of training) to build generalization capabilities
- Final phase: Minimal or no context (10% of training) to simulate real-world conditions with limited examples

By abstracting away the complexities of JSON-LD syntax and marketplace ontologies, the Offering Generator significantly reduces the technical barrier for data providers to participate in the SEDIMARK ecosystem, while ensuring semantic interoperability across all marketplace offerings.

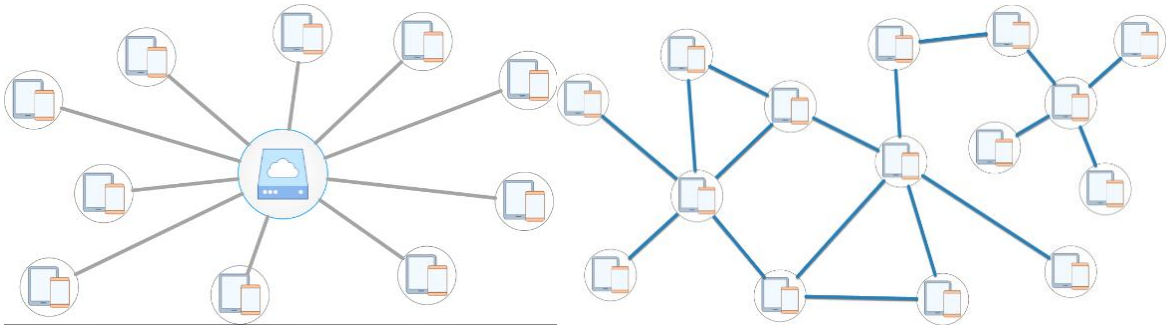
4.3 Distributed model training

Distributed model training within SEDIMARK can be divided into two main concepts:

1. **Federated learning (FL)**: this concept employs a server and a set of worker nodes. The role of the server is to orchestrate the overall training process through model aggregation and model parameter redistribution approaches. The role of the worker nodes is to train a local model based on the local data and the updated parameters received from the server. Here the server has a complete view of the worker nodes, while the individual worker nodes are only aware of the server.
2. **Gossip learning**: this concept only contains worker nodes. Here the worker nodes are connected with a subset of other worker nodes, where they share the model parameters. Through gossiping of model parameters between worker nodes all worker nodes in the network will eventually agree on a global model. Differently from the previous setting, here workers are aware of a subset of other workers, but generally, no worker has a full view of the complete network of workers.

The differences between federated and gossip learning are illustrated in Figure 37. The clear advantage of the Gossip approach is that it can avoid the single point of failure by eliminating the server from the computation. However, this comes at the expense of the gossip approach

taking longer to converge as it takes longer for the model updates to propagate through the communication network.



a) Federated Learning b) Gossip Learning

Figure 37: the difference between the architecture of a) federated and b) gossip learning.

Within SEDIMARK, two frameworks for distributed learning have been proposed and developed to cater for different scenarios and different user preferences.

- **deFLight**: this is a dynamic framework that is used for scenarios when data providers share through the marketplace either a model for training or a training process. This scenario is dynamic with participants being able to join or leave the training process at any given time.
- **Fleviden**: is an extensible tool to define computational graphs representing the FL agents and the operations therein. We put special emphasis on tools that improve interoperability at the AI/ML models level, acknowledging that not all data providers/sources will use the exact same software to train/run the models from a federated learning point of view

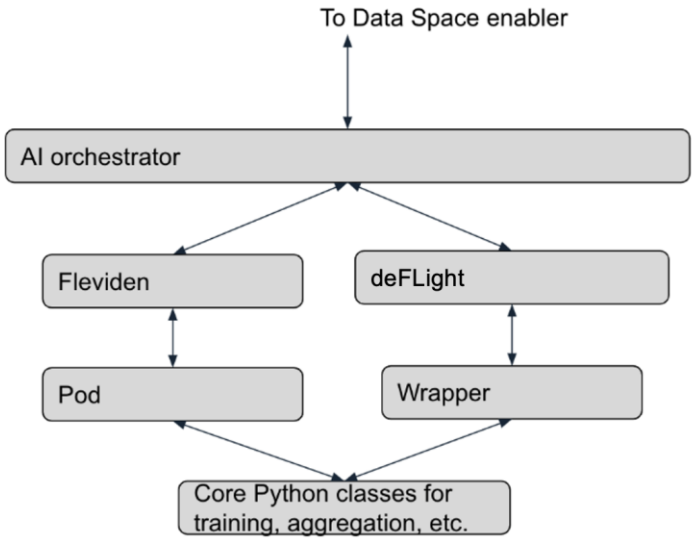


Figure 38: works for distributed learning developed within SEDIMARK.

As shown in Figure 38, the two frameworks are both designed to be modular and adaptive so that any project modules (i.e. models, aggregation mechanisms, privacy modules, etc.) can be developed in a framework agnostic way so that they can be used within both networks by exploiting their pods/wrappers.

When developing a FL solution, we need to consider the security requirements as a two-step process:

- The solution developer must develop the details of a federated algorithm by combining several techniques.
- The data provider must run a script representing the FL algorithm. This script must provide guarantees that it will follow the federated learning protocol strictly, e.g., keep the data and inference artifacts inside the provider infrastructure. This is what we call the minimum compliance requirement for any federated learning deployment.

There are several ways in which compliance can be accomplished. One alternative is that the data provider trusts the solution developer, which is not interesting from our perspective as in such a case we can fall back to more traditional machine learning solutions. Another approach is for the data provider to come up with their trusted review protocol to ensure the federated script provided by the solution developer is compliant. However, this is a difficult and expensive task that requires expertise and the capabilities to read through code and its dependency tree.

A proposal to be considered in future evolutions is the introduction of a third agent in the development process: the platform provider. The platform provider would create tools for the solution developer, such as automatic compliance checks, and demands the data provider to trust the platform provider but not the solution developer. This way, three-sided marketplace on top of federated learning:

- The data provider side, with private data and infrastructure offerings.
- The solution developer side, with novel algorithmic offerings and advanced implementations.
- The platform provider side, with their tooling offerings.

4.3.1 deFLight

SEDIMARK provides a flexible, fully decentralised model training framework. Titled “deFLight” this component offers a modular fully decentralised, asynchronous machine learning training solution. deFLight is built around a simple HTTP request/response architecture in order to conform with the constraints of the SEDIMARK connector. Additionally, deFLight is a dynamic framework, not requiring the set of training participants to be known ahead of time.

deFLight moves beyond a client/server model, and instead makes *nodes* the first class citizens within the distributed learning environment. While deFLight primarily targets fully decentralised model training, an FL paradigm can be easily created by arranging the nodes within a star topology. deFLight inspired by Flower, has been developed to be scalable, allowing participation of heterogeneous clients running on different platforms, be framework agnostic (the group of clients can use Tensorflow, PyTorch, etc. according to their group decision), etc.

Within SEDIMARK, a node can be instantiated by any participant wishing to collaboratively train a machine learning model and discover participants with similar compatible datasets. A training process, specifying a model architecture, a dataset specification, and a number of hyperparameters will then be advertised within the wider SEDIMARK trust infrastructure. New participants with compatible datasets can discover this advertised training process, and then launch their own deFLight nodes. Nodes follow a broader Gossip Learning (GL) protocol whereby they train locally their own version of the ML model and then send updates to other nodes that they select via a chosen sampling protocol. deFLight is developed in a modular approach, such that tools developed within the SEDIMARK project in other tasks, i.e., model

types, sampling strategies, quantization, etc. can be easily deployed within deFlight without the need of rebuilding those tools from scratch, but only with the use of simple wrappers.

4.3.1.1 deFLight node implementation

In many federated learning frameworks, there are two types of nodes that participate in the federated learning training process: (i) a client and (ii) as server. The client is the node that does the local training process, running the ML model on top of the local data, periodically sending the model parameters to the server and receiving the updated parameters for the next round. The server is the node that at each round samples the clients to run the local training, receives the parameters from the clients, runs the process for aggregating the parameters and sends the aggregated parameters back to the clients for the next round.

In deFlight the two types of nodes have been merged into a single deFlight node, whose internal structure is depicted in Figure 39 below. Inspired by the Gossip Learning approach where all clients are of a similar type, deFlight generalises the notion of a node in such a way that it can cover multiple distributed learning scenarios (as discussed below).

As depicted in Figure 39, a deFLight node consists of four main threads of operation:

- Receive thread, which handles the reception of weights from the rest of the nodes participating in the learning process. As discussed above, deFlight inherits from Flower the "communication-agnostic" feature, allowing multiple communication protocols between the nodes. However, currently, only HTTP is tested/supported, while in the future other protocols (i.e., gRPC) will be fully supported.
- Aggregation thread, which is the main thread that runs the sampling of the fellow nodes with which the node will communicate in the current round, and the runs rounds of aggregation of the parameter received from the fellow nodes.
- Training thread, which is the main thread that runs the local training process of the model based on the local data.
- Send thread, which takes the output weights from the training process and forwards them to the fellow nodes that participate in the current round.

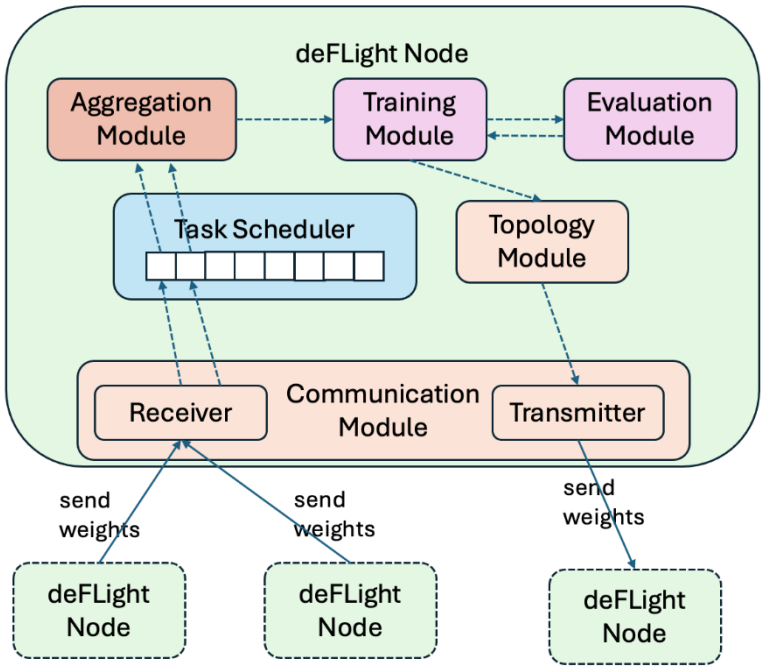


Figure 39: the internal structure of a deFLight node.

In its current state of development, deFlight uses a HTTP request/response communication protocol for sending and receiving model parameters, similar as within the Flower framework. The communication "receive" component hosts a Starlette HTTP server that continuously listens for incoming connections from other nodes, receives weights, and places them within a multiprocessing queue to be later aggregated. The use of a multiprocessing queue is critical to ensure that no weights are lost due to congestion, when receiving weights from many other nodes.

The main operational thread runs a local training process using Keras-Core for interoperability, such that the user can choose from either Pytorch, Tensorflow or Jax as their backend deep learning framework (more detail on model interoperability is given in SEDIMARK Deliverable D4.3).

Execution alternates between rounds of aggregating any model updates that have been collected in the aggregation queue, running the local training procedure, and then launching connections to communicate model updates to other nodes, selected via a sampling strategy.

The dynamic nature of the deFlight training process allows nodes to enter or leave the process at any given moment, without any special requirements, apart from following the SEDIMARK procedures for participating in the training as a service process.

4.3.1.2 Interaction of deFLight framework with the rest of the SEDIMARK components

To implement the distributed learning process, deFLight interacts with several other layers and components within the SEDIMARK architecture. These interactions are provided in the figures below.

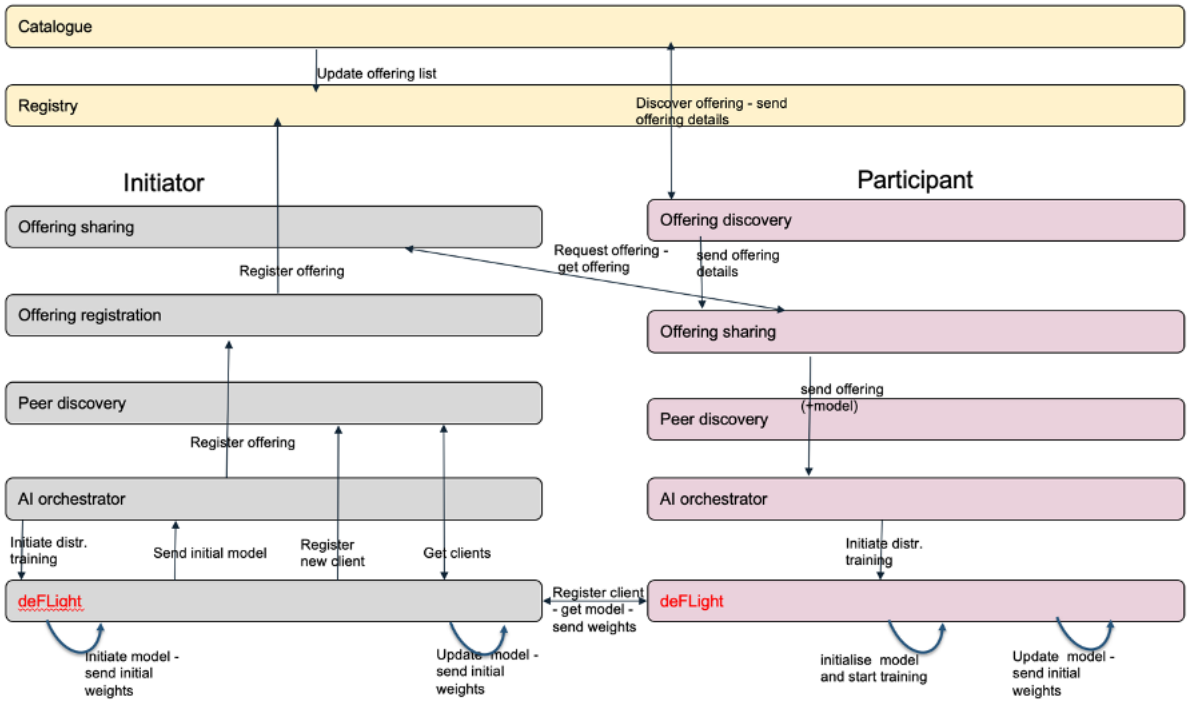


Figure 40: deFLight-based Federated Learning process.

Figure 40 shows the interactions between deFLight and the rest of the SEDIMARK components during the initiation and the execution of a Federated Learning process. It is clear that deFLight basically interacts with the AI Orchestrator, which is the main component that handles the training process. In this scenario, the assumption is that an “Initiator”, which is a SEDIMARK user (i.e. a provider) wants to start training a ML model on their data and then start a Federated Learning process, so that more participants join and help to train a better model. In this respect, the AI Orchestrator provides deFLight with the user preferences and settings, i.e. the framework to use, the model to train, etc. deFLight then initiates the training process, by initialising the model structure and its weights and forwards them to the Offering registration (through the AI Orchestrator) so that the training process is registered to the marketplace. A participant interested in the process can discover the training process in the marketplace, finding the respective offering and receiving it from the “Initiator” via the Offering sharing component. The details of the distributed process are forwarded to deFLight through the AI Orchestrator. The deFLight module on the Participant contacts the respective module of the Initiator to register as a client and receive the latest version of the model weights. Then deFLight initialises the local model and starts the local training process, updating the model and sharing the model updates with the “Initiator”.

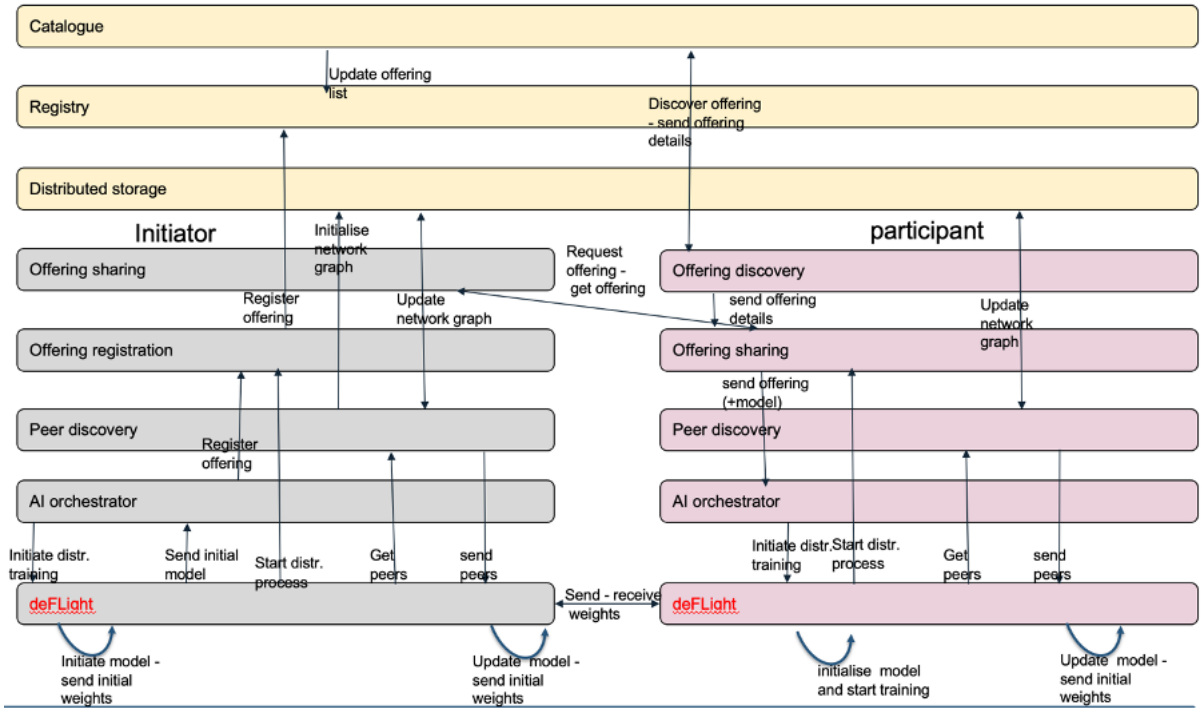


Figure 41: deFLight-based Gossip Learning process.

Figure 41 shows the interactions between deFLight and the rest of the SEDIMARK components during the initiation and the execution of a Gossip Learning process. The initialisation of the process is similar to that of Federated Learning (discussed above). The difference here is that there is not a server that holds a registry of the connected clients. To allow nodes to know the participants in the process, we exploit the distributed storage component of SEDIMARK, storing a "network graph", which is updated any time a node enters or leaves the training process. This is done by the "Peer discovery" component, which gets information from deFLight regarding the training process id, etc. When a "participant" discovers and wants to join the training process, the deFLight component initialises the received model and contacts the Peer discovery module to find out the fellow nodes participating in the current round. Then, deFLight executes the next round, training the local model, updating the weights and sending the weights to the fellow participants, while at the same time received the updates from its neighbours, performs the weight aggregation and continues to the next iteration of the training process.

In the current implementation, the communication between the deFLight nodes takes place directly through the deFLight component. In future versions, deFLight will be extended to use the interaction and communication protocol of SEDIMARK.

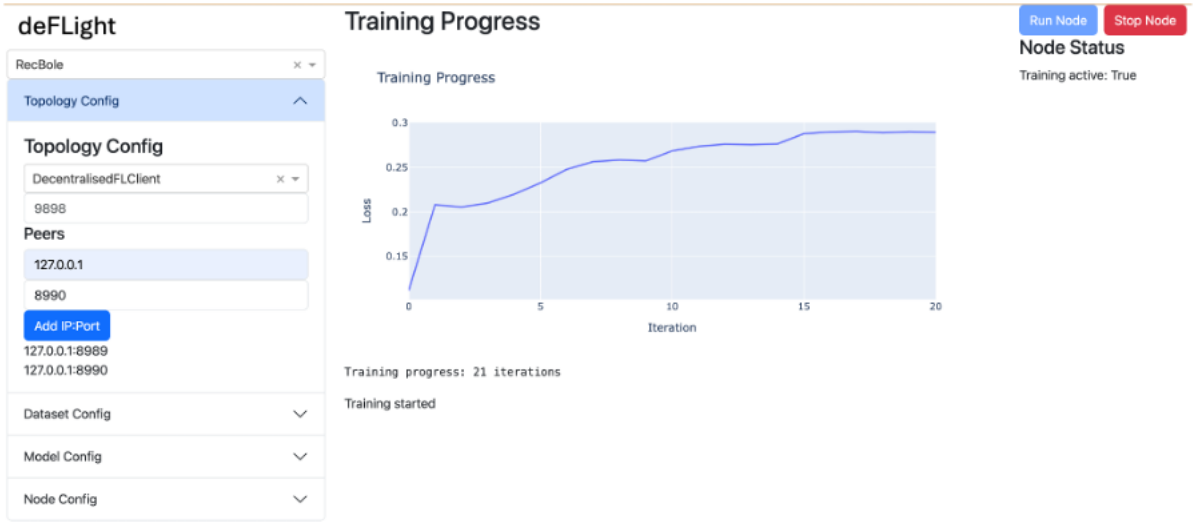


Figure 42: deFLight-sample user interface

deFLight is developed mainly as a command line tool, but for ease of use for novice users a simple user interface has been developed as seen in Figure 42. This user interface gives users the option to set the desired topology, to add their peers, to add the configuration about the dataset to use, the model to run, etc., and then they can start the local training process and/or participate in the decentralised training. When the training process starts, there is the option to visualise the results, showing the training process and the set metric, i.e. loss. A more thorough and SEDIMARK-specific user interface has been developed within the integration process and will be presented in WP5.

Beyond SEDIMARK, we will continue to iterate on the development of deFLight. We intend to test its robustness when run across a greater number of machines, with larger datasets and larger models. We intend to further modularise deFLight, to allow for the simple composition of the modules for aggregation, sampling and quantization that will be developed elsewhere within SEDIMARK. Some evaluation results are given in both D3.1 and D3.2, where the trade-offs between communication and performance are provided.

4.3.2 The Fleviden tool

As introduced in deliverable D3.3, Fleviden is an extensible framework developed by ATOS for orchestrating federated learning (FL) pipelines. Its architecture is based on the pipes and filters pattern, where agents (mainly several clients and one server) act as filters, and messages are exchanged through wires (pipes).

The basic functional unit in Fleviden is the pod, a modular entity with input and output wires. The mentioned pipelines are built by instantiating pods, linking them, and waiting for or bridging messages to/from external sources (e.g., HTTP or Kafka interfaces). Each pod encapsulates a logic that processes incoming messages and triggers outputs, enabling the creation of distributed learning workflows with privacy-preserving features. This modular architecture facilitates flexible and secure deployments across heterogeneous environments.

4.3.2.1 Handling of model interoperability in Fleviden

Fleviden includes support for the most used deep learning frameworks, such as Keras (with TensorFlow backend), Torch, and ONNX, enabling a wide range of neural models to be

incorporated into federated learning pipelines. Fleviden pods were designed to load and run serialized model definitions, such as Keras .h5, TorchScript .pt, or ONNX .onnx files.

Since then, Fleviden's capabilities in terms of model interoperability have expanded significantly. One major update is the inclusion of native support for models developed using scikit-learn, a widely adopted machine learning library. Notably, support has been added for decision trees, a model type not previously available in Fleviden. This addition expands Fleviden's use cases, particularly in areas where interpretability, computational efficiency, and non-gradient-based training matter.

Moreover, a new feature has been added to further enhance model flexibility: in addition to loading models from files, Fleviden now allows creating models directly from Python class definitions. That is, an architecture can be defined directly in a .py script and injected into the Fleviden trainer pod at runtime. This provides a smoother development experience, enabling users to define and initialize models programmatically within their application code. This functionality is particularly useful in research settings or when deploying experimental models without going through a full serialization pipeline.

Fleviden also has a variety of customizable pods that allows developers to define the internal logic that gets executed when calling its interfaces, which is usually encapsulated and hidden from end-users. In this regard, the CustomTrainer pod –and other Custom pods from other packages– can be leveraged to define a fully customized training pipeline that is adjusted to the use-case at hand, which enables training models in frameworks that are not natively supported within Fleviden, as long as the required libraries and dependencies are installed in the application environment. All user-defined custom functions must follow the Fleviden request message convention, namely, by taking a Python dictionary as input and returning a JSON-serializable dictionary as output. By enforcing this convention, we make sure that Fleviden is open for extension without breaking interoperability with the rest of pods.

These developments not only increase the scope of Fleviden's interoperability layer but also align with SEDIMARK's broader objective of supporting heterogeneous federated learning across multiple organizations, technologies, and regulatory requirements.

4.3.2.2 Handling of service interoperability in Fleviden

In deliverable SEDIMARK_D3.3, working on the development of a high-level scripting layer for Fleviden, referred to as Fleviscript, was reported. The initial motivation behind this proposal was twofold:

- Technical simplification: Fleviscript was designed to reduce the complexity of programming federated learning workflows directly in Python. It aimed to provide a more user-friendly interface for non-expert users or stakeholders who did not need fine-grained control over pod-level logic but still required the ability to define orchestrated federated learning pipelines.
- Compliance and security: This script also aimed to satisfy a key business requirement: to ensure that client scripts remained compliant with federated learning principles. This meant enforcing local data processing and limiting the set of operations to those that could not compromise privacy (e.g., create, link, wait, bridge), thereby reducing the attack surface and ensuring minimum compliance guarantees.

Fleviscript was conceived as a declarative, domain-specific language design to connect Fleviden pods and build federated learning workflows. A full specification of its syntax and semantic model (including the structure of import, input/output wire registration, variable

assignments, and pod configuration instructions) was detailed in an annex of deliverable SEDIMARK_D3.3.

However, as the development of the Fleviden framework progressed and implementation of some use cases and more advanced orchestration logic began, it became clear that maintaining a separate domain-specific scripting language posed several challenges:

- Limited flexibility and capabilities: Although Fleviscript was originally meant for simple workflows, it didn't have enough power to handle more dynamic or complex ones. Adding features like conditions, loops, or error handling would have made the interpreter much more complicated and moved it away from its goal of being a safe and straightforward language.
- High maintenance cost: Introducing a new language also meant maintaining a full set of tools, including a parser, interpreter, debugger, and possibly even custom editors or validators. This added significant technical work without offering clear long-term advantages compared to using existing Python-based tools.
- New needs and use cases: As the Fleviden ecosystem grew and new partners joined, the need for flexibility and easy integration became more important than keeping the language simple. Supporting different environments, performance improvements, and integration with external connectors required a more flexible and extensible setup.

In this context, the decision was made to deprecate the Fleviscript approach and replace it with a more modular and scalable system based on multiple layers. This new design allows developers and end users to choose the level of control or simplicity that works best for them. Fleviden now uses a layered architecture that offers different ways to define federated learning pipelines, depending on the user's experience the complexity of the deployment. Currently, the two main layers are core and engine.

The core layer represents the lowest level of abstraction within the Fleviden framework. It is designed for expert users who require full control over the definition and execution of federated learning workflows. At this level, users interact directly with pods, which are processing units organized by function, such as aggregators (*fleviden.core.aggregators*), trainers (*fleviden.core.trainers*), and others. Developers can create custom pods by using the *fleviden.core.pod.pod.Pod* class, enabling fine-grained customization of the logic involved in the federated learning process. This layer is ideal for those with a deep understanding of federated learning principles who need to implement novel or non-standard patterns.

Core pods are simple by definition, implementing an atomic functionality with one specific responsibility. Different Fleviden packages follow a strategy pattern approach, making pods with similar logic easily interchangeable. For example, all aggregators register the same input and output wires so that the specific aggregation technique (weighted average, median, Krum, etc) can be selected at runtime. To enforce this strategy pattern approach, although Python supports duck typing, Fleviden relies on the ABC (Abstract Base Classes) module to define the base pods of each package (e.g., an abstract Aggregator). This design choice is key to building generic pipelines and enabling a programming-to-interface approach that facilitates building Fleviden's layers of abstraction.

The engine layer, by contrast, provides an intermediate abstraction that significantly simplifies the construction of federated learning systems. It offers a collection of high-level components that encapsulate common FL workflow patterns, such as the local data loading or the server's aggregation setup, which would require several core pods connections to achieve. Engine level components are built as Fleviden pods that are internally composed of core level atomic pods,

mixed and matched with common connections that abstract away the links that a typical Federated Learning pipeline would implement. For example, the application script of a Server agent is typically comprised of an aggregation pipeline (i.e., gathering the local updates from clients, performing the aggregation step and updating the global model) and an orchestration block (handling the subscription of clients, selecting the active participants in each round, and managing the start and end of the process). The set of pods and the connection that implement these pipelines form a common pattern that can be encapsulated in an engine-level Server package to simplify greatly how the Fleviden application is deployed. By exposing the configuration parameters of the core pods, but treating their connections as a black-box, users maintain the flexibility to tune the process to their particular needs without having to struggle with defining the computational graph at the lowest level.

It's important to note that engine pods do not implement any additional logic besides what is already present in the core layer pods they encapsulate. Instead, they simplify the definition of the computational graph and abstract the configuration of hyperparameters for common flows. In the application script for Fleviden agents, users can connect engine and core pods without distinction, as there are some core components that are feature complete and thus do not implement an analogous engine variant (e.g., communication protocols).

In conclusion, while the original concept of Fleviscript addressed important goals such as simplicity and compliance, the framework's technical requirements evolved over time, making a change in direction necessary. The new multi-layer design in Fleviden offers greater flexibility, allowing users to choose between more control or a higher level of abstraction, depending on their needs. This shift improves compatibility with a wider range of use cases and makes future development easier to manage. It reflects a deeper understanding of what is needed to support federated learning across different environments, and positions Fleviden as a reliable framework for integration within SEDIMARK and other contexts.

5 The DLT Infrastructure

This section describes the functionalities and the final architectures of the DLT Infrastructure employed as the underlying foundation for the final version of the SEDIMARK Marketplace.

The SEDIMARK Marketplace relies on a robust DLT Infrastructure to underpin its operations. This foundation provides verifiable and immutable records of Participant information and Offering details, ensuring trustworthiness and non-repudiation.

The target of the infrastructure is identified in the following functionalities:

- Identity Management: Securely manages Participants' identities.
- Metadata Management: Enables information related to Offerings for the Catalogue.
- Trust Metadata Storage: Provides transparent and auditable records of trust information.
- Tokenization: Facilitates secure asset ownership and trading.
- User Wallet Integration: Seamless integration with participant wallets for efficient transaction management.

Such infrastructure enables interconnection among the various enablers and mechanism realising the functionalities of the SEDIMARK Marketplace.

The DLT infrastructure is built on a two-layer architecture:

1. Layer 1 (L1): IOTA Tangle, offering decentralized data sharing.
2. Layer 2 (L2): IOTA Smart Contract (ISC) chain, enabling smart contract execution and transactions.

These two layers have been previously analysed in the SEDIMARK deliverable D4.1. The initial version of the hardware and software implementation of this infrastructure, described in SEDIMARK deliverable D3.3, has been adopted for experimenting and testing of the various features developed during the SEDIMARK project.

The underlying SEDIMARK infrastructure has been improved and extended to reach the final and stable version. The following subsections analyse instead the final implementation of the underlying infrastructure from the hardware and software point of view.

5.1 Background and recap

The IOTA network operates on a distributed ledger system, known as the Tangle. This technology forms the core of the SEDIMARK decentralized platform by providing a secure and efficient way to record transactions. The Nodes within the network are interconnected and maintain this shared ledger through a consensus protocol.

The IOTA full-node software is called HORNET and it is written in Go. HORNET serves as the foundation of any IOTA node network configuration. This software is designed to ensure efficient operations and flexibility in deployment.

One of the key features of HORNET is the “INX” interface, which allows seamless extension through ad-hoc plugins. This allows developers to create specialized functionalities tailored to their specific needs, opening up a world of possibilities.

HORNET additionally enables the integrated dashboard, which provides real-time transaction information for monitoring, allowing users to visualize the activity on their node in real time.

To enhance the capabilities of IOTA beyond basic transactions, they introduced Wasp - a smart contract platform built upon the IOTA network. Wasp uses its lightweight and efficient core contracts (EVM – Ethereum VM) to handle EVM/Solidity contracts (as well as Wasm based contracts) – expanding the application of the IOTA network. This allows developers to build complex decentralized applications with ease.

To ensure seamless integration with existing tools, Wasp provides a standardized JSON-RPC service for user interaction. Users can effortlessly connect their wallets or development frameworks to interact with the EVM layer through this interface. Deploying EVM contracts on Wasp becomes as simple as connecting your development tool to this defined endpoint.

In essence, IOTA's technology combines a distributed ledger system, decentralized nodes, and specialized software for smart contract development, empowering developers to create innovative solutions across a variety of domains.

5.2 Final architecture

The final version of the infrastructure employed in the SEDIMARK Marketplace is composed of a layered architecture. From a high-level point-of-view, it is composed with the two distinct DLT layers stacked each other, i.e., IOTA Tangle (Layer 1) and IOTA Smart Contract Chain (Layer 2). Such layers are stacked on top of the hardware infrastructure providing the necessary computational capabilities.

A simple installation of the software stack prepares a node able to interface and connect with the public network (i.e., the mainnet) of the IOTA Foundation. As a consequence, an instance of a Hornet node would be consistent with the content of the ledger public network, holding data and transactions not related to SEDIMARK Marketplace. Also, the computational capabilities of the hardware acting as nodes and their related cost would be exploited to become a peer of the decentralized public network.

For the scope of the SEDIMARK project, the underlying structure is reserved and adapted for the project target. Thus, the SEDIMARK Marketplace has its root in a private instance of the entire DLT, as well as the necessary smart contracts engine.

The official repository containing the resources for the instantiation of the infrastructure is https://github.com/Sedimark/hornet-extra/ .

The collection of configurations and scripts enables both to join an existing decentralized SEDIMARK Marketplace and to instantiate a new existing infrastructure, with minimal intervention and changes needed. The external dependencies are also limited and managed resorting to containers technology for ease of deployment and portability.

The top-level underlying infrastructure of the DLT is reported in Figure 43. The figure represents the two logical layers previously defined, together with the auxiliary services block. The logical intra- and inter- communications for the two layers is described in SEDIMARK deliverables D4.1 [21] and D4.2 [22].

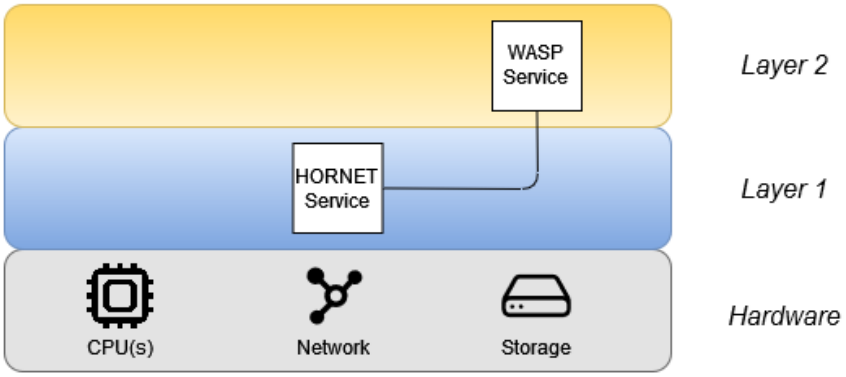


Figure 43: Layered architecture for DLT infrastructure

From an architectural point-of-view, the changes of the infrastructure in the second part of the project did not directly concern the two specific layers. Instead, they mostly focused on the scalability of the infrastructure, interconnection and management aspects. As example, in the previous settings, the various instances of Hornet nodes were not directly exposed to the internet. In the current and final version, they are instead able to communicate directly with external services, enhancing the connectivity in a real decentralized manner. These features requested additional software and the related configuration, not specifically needed by the logical infrastructure (L1 and L2) but required to allow proper interconnections.

Additional specific services have been added and newly created. Such services enable a set of convenient functionalities needed to limit manual intervention. For instance, the services for sharing the current status of the ledger (i.e., the so-called *snapshots*) allow other Nodes to connect more rapidly and in a programmed manner.

Other instantiated services instead focused on the management side. Ad-hoc software has been integrated in the previous software stack to provide the owner of the Node a quick and simplified interface to get an idea of the current state of the situation at a glance. Obviously, such situation might in turn refer to different aspects of the management. For example, the situation to glimpse might refer to the logical layers (both on ledger and chain), which is provided through the dashboard services instantiated and already configured.

5.3 Architecture Components

From the logical point-of-view each node of the underlying infrastructure is made of several components. The minimum set of components needed to deploy the infrastructure is formed with Hornet and Wasp software, as in the previous version.

The current and final set of components in shown in Figure 44. Each component has a specific role that either define a functionality or complements it.

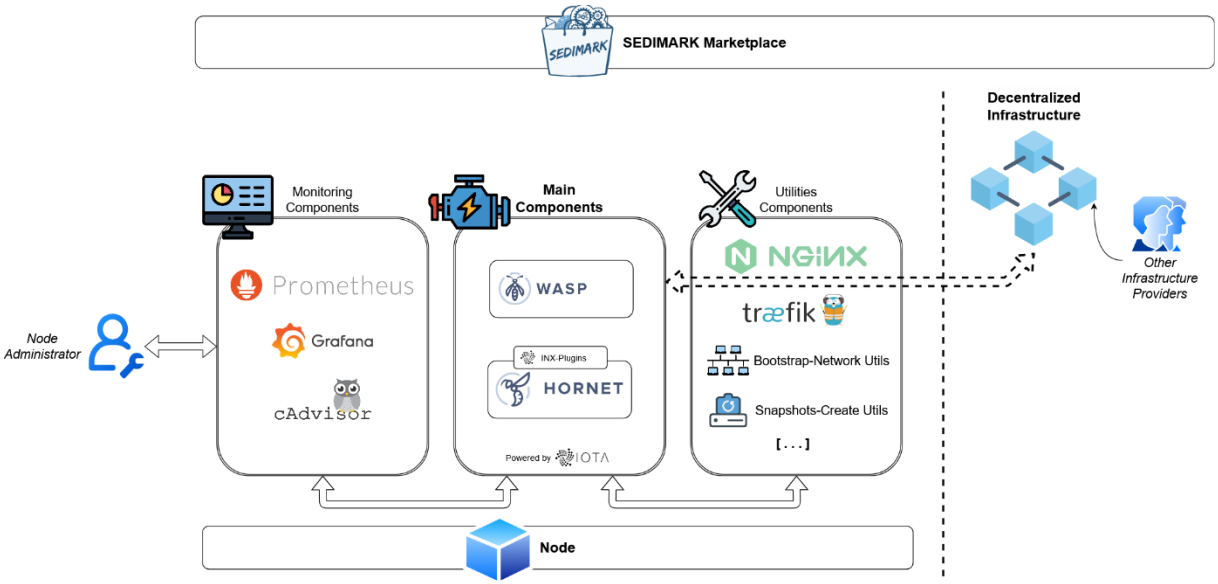


Figure 44: Architectural components of the infrastructure

The figure above shows the architectural components of the infrastructure and how they are interconnected.

The SEDIMARK baseline infrastructure relies entirely onto the IOTA ecosystem, which is powered by two essential components (*Main Components*). Such components are modular by design and their functionality is extended through the INX-Plugins.

Main Components

- Hornet: Software for L1. It is lightweight and efficient IOTA node implementation designed for high performance and scalability. Hornet is built in Go and focuses on providing a robust and user-friendly interface for interacting with the IOTA Tangle. It supports various features such as automatic peer discovery, a built-in API for transactions, and integration with INX plugins, making it suitable for both developers and end-users.
- Wasp: Software for L2. It is smart contract platform for IOTA that enables the development and execution of decentralized applications (so-called dApps). Wasp allows developers to create and deploy smart contracts using familiar programming languages, providing a secure and scalable environment for executing complex logic on the IOTA Tangle. It features a unique consensus mechanism and supports various functionalities, including state management and event handling, to facilitate the development of innovative applications.

One of the key strengths of Hornet is its extensibility through INX plugins, which enhance its functionality and provide additional capabilities. Below is reported the list of the INX plugins integrated with the deployment of Hornet, detailing their specific purposes.

INX-Plugins enabled

- INX-Dashboard: provides a user-friendly interface to overview the system.
- INX-MQTT: provides an event-based real-time streaming node API. The built-in MQTT broker offers a list of topics clients can subscribe to, to receive the latest blocks and outputs attached to the tangle.

- INX-Indexer: is an indexing tool to provide structured data that can be searched and utilised by wallets and other applications. The indexer maintains its own database separate from that of the node.
- INX-Participation: is an extension for nodes to enable on-tangle voting. The extensions maintain its own database separate from that of the node and provides means to track events and votes.
- INX-POI: enables you to generate and verify Proof-of-Inclusion of blocks in the Tangle. Given a piece of data or transaction and the proof, it is possible to verify whether it was included in the Tangle at any given time.
- INX-Coordinator: performs the core functionalities of a node. It generates and issues the Milestones transactions, which are a special kind of transactions employed as markers of the progress and for providing timestamps for different points in the Tangle. Any transaction points, directly or indirectly, to at least one Milestone. The coordinator decides which transactions to approve. Moreover, it prevents double-spending issues and ensures that transactions cannot be reversed. The coordinator helps new nodes join the decentralized network by providing checkpoints for history, promoting faster synchronization. This ensures that new nodes have a starting point for validating the Tangle.
- INX-Faucet: faucet is employed for dispensing native tokens. For development and testing purposes, two faucets are deployed respectively in L1 and L2. (Note that L2 faucet is not an INX plugin. Conversely L1 faucet is the INX plugin).
- INX-Spammer: is a client application, running locally, which sends dummy transactions to the Tangle to provide a constant flow of transactions. This happens for performance reasons: a new transaction must be indeed referenced by at least three blocks. The spammer transactions increase the reference and confirmation rates of the DLT.

Each node is complemented with a set of components that enables the real-time observation of transactions and smart contracts. Additionally, effective monitoring is crucial for maintaining the health and performance of the SEDIMARK infrastructure. The monitoring components provide real-time insights on resource usage and application performance. Such components are listed below.

Monitoring Components

- cAdvisor (Container Advisor): collects, aggregates, and exports metrics about container resource usage and performance characteristics, such as CPU, memory, and network I/O
- Prometheus: is a monitoring and alerting toolkit designed for reliability and scalability. It collects metrics from configured targets at specified intervals, stores them in a time-series database, and provides a powerful query language for analysis. Prometheus is particularly well-suited for monitoring microservices and cloud-native applications.
- Grafana: is an analytics and monitoring platform that integrates with various data sources, including Prometheus. It provides a rich visualization layer for displaying metrics and logs through customizable dashboards. Grafana allows users to create interactive graphs and alerts, making it easier to monitor system performance and health in real-time.

Utility and services components play a vital role in supporting the IOTA infrastructure by providing essential functionalities such as traffic management, load balancing, and custom service integrations. These components enhance the overall architecture, streamline

operations, and facilitate the deployment of additional features, ensuring a robust and efficient environment for both developers and users.

Additional Utility and Services Components

- Traefik: reverse proxy and load balancer designed for microservices. The configurations set the services and routes traffic to them based on the routes defined. Traefik supports various backends, including Docker, Kubernetes, and more, and provides features such as SSL termination, traffic management, and real-time monitoring through a user-friendly dashboard.
- Nginx Proxy Manager: nginx with a graphical interface for ease of configuration.
- Create-snapshots: creates an initial (empty) snapshot.
- Bootstrap-Network: creates the file needed to start a new DLT.

5.4 Physical Architecture

The physical architecture is employed to provide computational capabilities to the software components described in the previous section. The two layers (L1 and L2) are mapped onto physical hardware.

The decentralized network for L1 is composed by four instances of Hornet. A copy of the software for the node is deployed across different physical machine. The functionality of each Hornet node is extended with the respective INX-Plugins. For every instance of Hornet, a corresponding instance of Wasp (for L2) is instantiated as well on the same machine. Each Hornet node is interconnected with the others. Analogously, each Wasp instance is interfaced with both other instances the related instance of Hornet. The physical setup is shown in Figure 45.

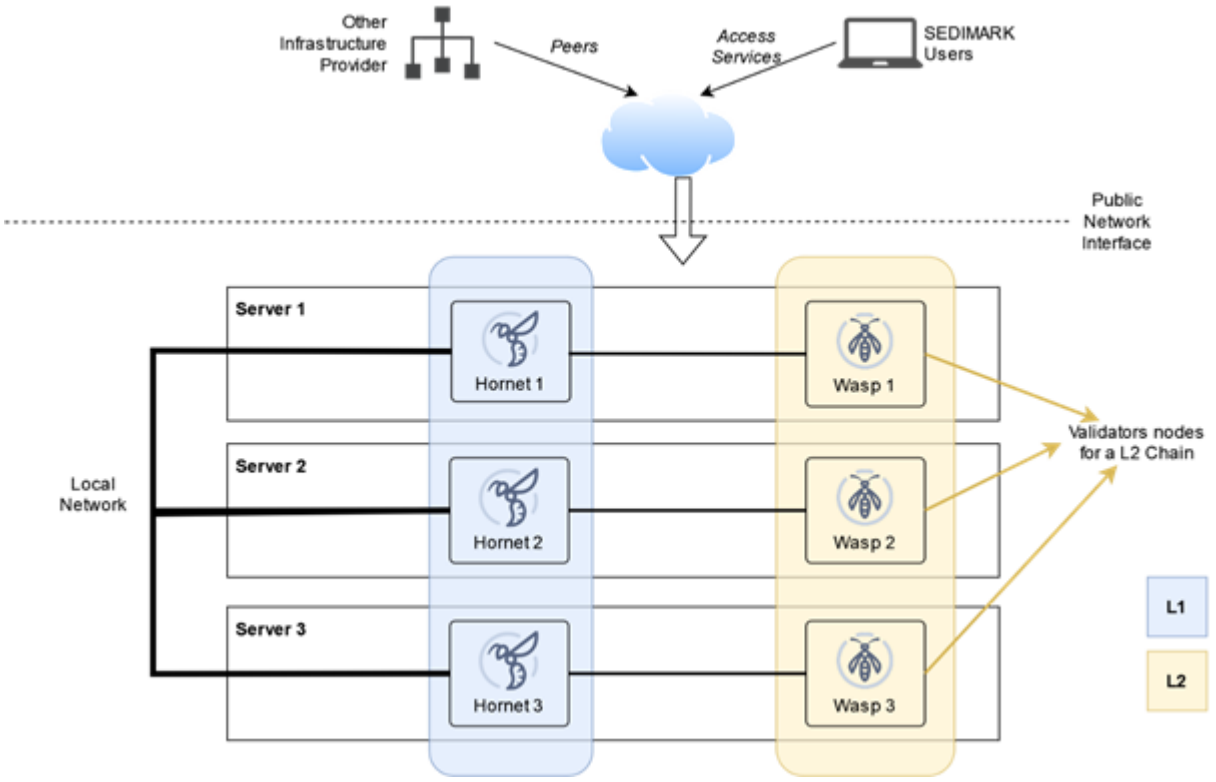


Figure 45: instantiation of DLT Layers onto physical hardware machines

The physical deployment exploits servers installed on the partner premises (Fondazione LINKS – LINKS, University of Cantabria – UC and EGM) for the duration of the project. As example, the physical machine in LINKS are three Dell server blades (R650), each equipped with Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz (with 16 cores). The RAM available amounts to 64GB (7% is used). The disk space available is 900GB and the rough total space taken up by the installation with the operating system is about 8%. The Tangle currently employ 30GB. However, it does not have a fixed upper limit.

Already in the first instantiation described in SEDIMARK deliverable D3.3 [23], the servers were interconnected to each other. Moreover, one additional server had a public interface exposed to the internet with a Public IP address. This allows the specific machine to act as a gateway for connecting other decentralized external nodes.

Conversely from the first version, every server in the final instance is able to communicate with other nodes, either internal or external. In turn, external clients are able to interact with the network while maintaining secure communication among the instances. The specific public interface is preferred for ease of communication and for scalability purposes. The server with this interface is in fact equipped with a reverse proxy and load balancer able to handle multiple concurrent requests. From the security point-of-view, the services exposed resort to HTTPS communications, implying the necessary actions to undertake for obtaining and keeping updated certificates for establishing secure connections.

In the context of trust, the initial setup provided multiple Wasp nodes, but a single validator. In the final version, all the validators are reachable from outside peers enhancing the mutual trust required for the validation of transactions. Moreover, such architectural changes to the connectivity enable a good degree of decentralization and node distribution of the network.

In the final version of the infrastructure, several endpoints have been defined and published for the sake of improving access. Here below it is reported a complete list of the endpoints with their intended usage.

List of public endpoints:

https://stardust.linksfoundation.com/node1/dashboard/

https://stardust.unican.sedimark.eu/dashboard/

Hornet Dashboard Public page

https://stardust.linksfoundation.com/node1/wasp/dashboard/login

https://stardust.unican.sedimark.eu/wasp/dashboard/

Wasp Dashboard Public page

http://192.168.94.12/grafana/login

Grafana Webpage – private monitoring

http://192.168.94.12:8088/dashboard/#/

Traefik Webpage – private monitoring

http://192.168.94.14:81/login

Nginx Proxy Manager Dashboard – Configuration of nginx

https://snapshots.stardust.linksfoundation.com/l1/

Hornet snapshots download page (delta and full)

https://snapshots.stardust.linksfoundation.com/l2/

Wasp snapshots download page

https://stardust.linksfoundation.com/node1/sedimark-chain

https://stardust.unican.sedimark.eu/sedimark-chain

Node 1 – endpoint for accessing SEDIMARK RPC (Node 2 and Node 3 also available)

https://stardust.linksfoundation.com/node1/api/routes

Node 1 – endpoint for accessing APIs (Node 2 and Node 3 also available)

https://stardust.linksfoundation.com/faucet/l1/

L1 Network Faucet

https://stardust.linksfoundation.com/faucet/l2/

L2 Network Faucet

https://json-rpc.evm.stardust.linksfoundation.com/sedimark-chain

Endpoint for accessing SEDIMARK RPC (this endpoint is load-balanced with round-robin logic)

https://stardust.linksfoundation.com/node1/wasp/api/routes

Node 1 – Endpoint for accessing the Wasp API (Node 2 and Node 3 are also available)

The servers are interconnected to each other in a local network. These three machines are the peers composing the DLT and the Smart Contract chains for the SEDIMARK Marketplace. Incoming connections related to the digital identity are managed at L1 level, where the transactions store (partially) the elements of the SSI. Smart contract applications are deployed

at L2 with the ISC allowing the trading of assets between SEDIMARK users and implementing the Marketplace business logic.

The infrastructure exposes a public interface that allows the interactions with remote users. SEDIMARK users are able to connect and interact with the services detailed resorting to the toolbox and the applications developed during the other WPs. The partners who want to enforce the capability of the SEDIMARK Marketplace can provide their own computational capabilities and storage facilities by deploying their own instances. The software stack is containerised for the ease of deployment on an external physical infrastructure. A newly deployed infrastructure can be linked to the existing one, thereby extending the capability of the whole system. In such a way, the partners' infrastructures become members of the ledger by acting as peers of the decentralized network and/or of network of validators.

5.5 Scalability considerations

The final SEDIMARK infrastructure is designed with scalability and resilience in mind, using the hardware composed of three interconnected physical on the partners premises to host Hornet and Wasp instances. This setup enhances the performances and also demonstrates that the system can grow as demand increases.

The decentralized nature of this infrastructure is a significant advantage, as it allows additional servers to join the network easily. This flexibility means that other users (usually the *Providers* in the SEDIMARK Marketplace, but also the Partners of the SEDIMARK Consortium) can contribute their own nodes, further enhancing the robustness and capacity of the overall system. The infrastructure as-is at the time of writing this deliverable is already resilient to server failures and maintenance activities considering the number of servers employed. In this configuration, if one server experiences issues or requires maintenance (meaning downtime due to e.g., physical updates of hardware), the remaining servers can continue to operate, ensuring uninterrupted service. Therefore, this decentralized architecture fully support scalability and fosters also external participation from other partners and user willing to strengthen the infrastructure.

Additional mechanisms have been established to facilitate the scalability of the infrastructure. The RPC endpoint is load-balanced according to a round-robin scheme, providing the users and other services a simplified access, through load-balancing. Nevertheless, it is still preserved the possibility to access directly the RPC of the L2 chain of the specific node.

6 The Storage enabler

In the digital age, data stands as one of most important components of any modern business ecosystem. Its value is especially magnified in the realm of distributed marketplaces, which serve as hubs of vast and diverse data exchanges across various regions.

These systems go beyond traditional storage paradigms by spreading data across multiple physical locations, be it within a single data centre location or across countries. Such an approach is not just a matter of scalability, but a pivotal strategy to ensure data availability, fault tolerance, and efficient distribution.

As these platforms deal with heterogeneous data – from city traffic information and user profiles to transaction records and user-generated content – the need for a robust, scalable, and interoperable storage mechanism becomes implicit.

Furthermore, as AI and machine learning continue to play a more significant role in data analysis and decision-making processes within these marketplaces, the integration between storage and computational resources gains even more prominence.

6.1 Significance of data storage

There are three pivotal attributes one must consider when choosing the data storage solution for these systems: scalability, fault tolerance and data interoperability.

- Scalability refers to the system's ability to handle increased load or demand by adding more resources or nodes, without affecting the system's performance or architecture. Distributed storage systems, unlike traditional systems, don't require massive fine-tuning or downtime to scale. As the need arises, new storage nodes can be incorporated seamlessly.
- Fault tolerance is the property that enables a system to continue operating seamlessly in the event of the failure of some of its components. Distributed storage systems typically replicate data across multiple nodes. This means if one node encounters a failure, the system can retrieve the data from another node. This redundancy always ensures data availability.
- Data interoperability is the ability of systems and services that create, store, and exchange data to have clear, shared expectations for the contents, context, and meaning of that data. In a distributed marketplace, data might originate from various sources - different vendors, platforms, or services. Distributed storage solutions can store diverse data types and structures, offering a unified access point irrespective of the data's origin. For marketplaces that involve multiple stakeholders, from vendors to third-party service providers, data interoperability ensures that all parties can access and understand the shared data, facilitating smoother collaborations and transactions.

6.2 Storage Enabling software

Below are defined the storage enabling software split in two sections, one for data storage, using NGSI-LD broke, and the other for model storage, using Minio.

6.2.1 AI model storage enabler (Minio)

Minio [24] is a high-performance, distributed object storage server, designed for large-scale data infrastructures. It is S3 compatible, built for the cloud-native world, supports object versioning, encryption, and event notifications. Minio provides scalable storage for marketplace assets, in the case of SEDIMARK marketplace it will be used as a performant storage enable for model storage through MLFlow, which is a model management software, and it will use Minio to store the models created inside the SEDIMARK Toolbox.

Each toolbox will have deployed an instance of Minio to save and load models through MLFlow and to pass the models to the connector when it is required to retrieve a model.

The deployment for Minio will be done through a docker compose file that will deploy the entire toolbox with all the necessary components for a provider or consumer to be able to create, share and retrieve models. The deployment for the SEDIMARK Toolbox will be available on GitHub to ease the installation of the Toolbox.

For a consumer to retrieve a particular model stored at a provider, the connectors of both the provider and the consumer needs to interact with Minio, either directly or indirectly, in such a way that the connector at the provider side will get the model through a REST API from Minio and pass it to the connector on the consumer side.

Below is described the components that are involved in the transaction of a model from a provider to a consumer, and the process behind it.

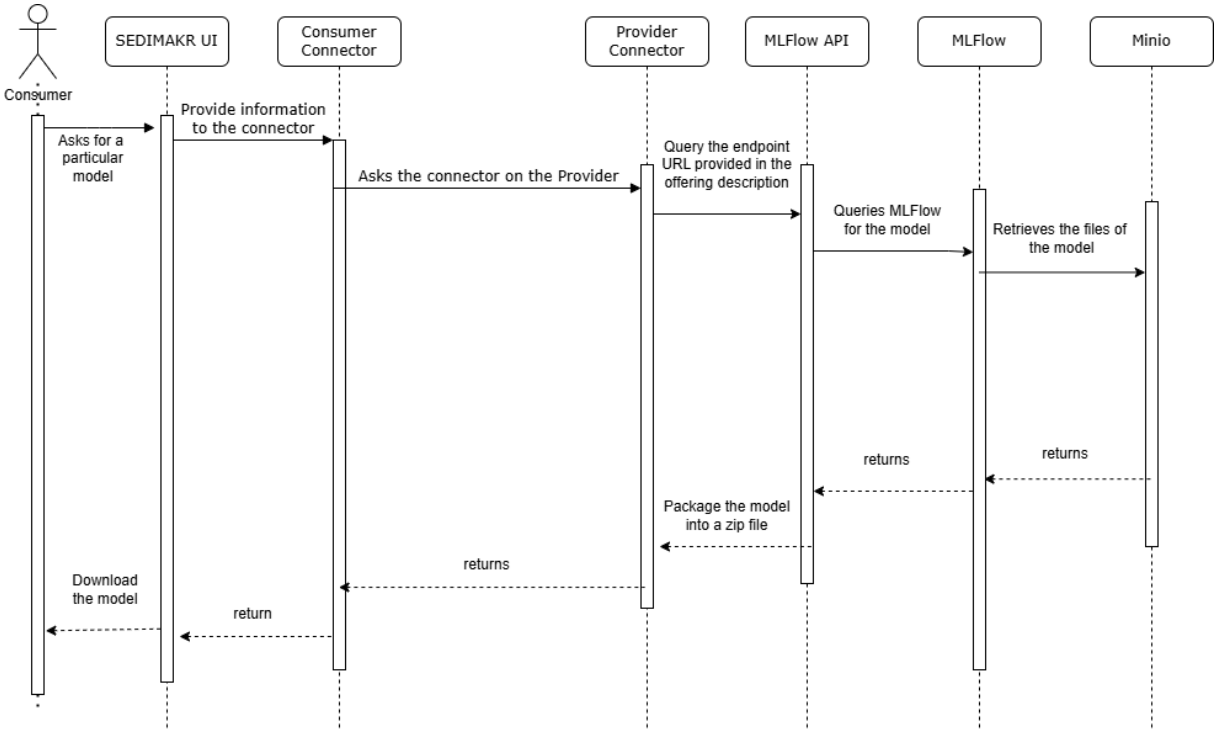


Figure 46: Sequence diagram for model downloading from the consumer side.

All the components presented in the diagram are present on GitHub.

6.2.2 Data storage enabler (NGSI-LD brokers)

NGSI-LD brokers implementing the temporal API described in section 2.2 also act as a storage. As an example, the Stellio context broker embeds a PostgreSQL database empowered with

TimescaleDB and PostGIS extensions to handle time series as well as geographic information. Data exchange within the Stellio broker are made over a high-speed exchange bus built on Apache Kafla which allow to scale while a spring boot-based API gateway ensures the conformity to the NGSI-LD specification. Such an implementation provides interoperability while allowing fast ingestion rates. As visible in Figure 47, ingestion rates of more than 20k events/s have been demonstrated on machine with 8 vcore, 32 Go RAM and 4 To disks. Based on NGSI-LD specification, deployment architecture includes centralised, distributed and federated options.

The Stellio broker has been significantly extended to include context source registration capability, enabling it to participate in distributed deployments. This enhancement allows Stellio to support multiple deployment configurations beyond its original centralized architecture, including distributed and federated deployments where context sources can register themselves with information they can provide on request. In distributed settings, Stellio can discover context sources that may have information for answering requests based on their registrations, request and aggregate information from different context sources, and provide it to requesting context consumers. The broker's architecture is built around a modular design following reactive and functional paradigms, with services that are thoroughly tested and deployed in many production environments.

Based on NGSI-LD specification, deployment architecture includes centralised, distributed and federated options. The centralized architecture features a central context broker that stores context information provided by context producers, while distributed settings allow all context information to be stored by context sources themselves. In federated architectures, context sources can be context brokers that make aggregated information from lower hierarchy levels available, and these architectural approaches are not mutually exclusive - actual deployments may combine them in different ways. The deployed architecture is designed to evolve from centralized to distributed to federated configurations without requiring software reinstallation.

Figure 47: example of scaling capacity of Stellio context broker (number of inserted items per second over time)

6.2.3 Offering storage enabler (Catalogue)

Offerings are modelled in RDF and formatted in JSON-LD, and they can be found either in the participant premises (as part of a Participant Self-Listing) or in the Marketplace Catalogue.

Offerings embedded in Self-Listings are stored in a local relational database based on PostgreSQL (see Figure 48). This storage solution is shared by other components in a

SEDIMARK Toolbox (e.g. DLT-Booth, Stellio Context Broker). The component in charge of storing and maintain the Self-listing, the Offering Manager, stores as well a reference to the specific Offering and its hash into the DLT, through the DLT-Booth.

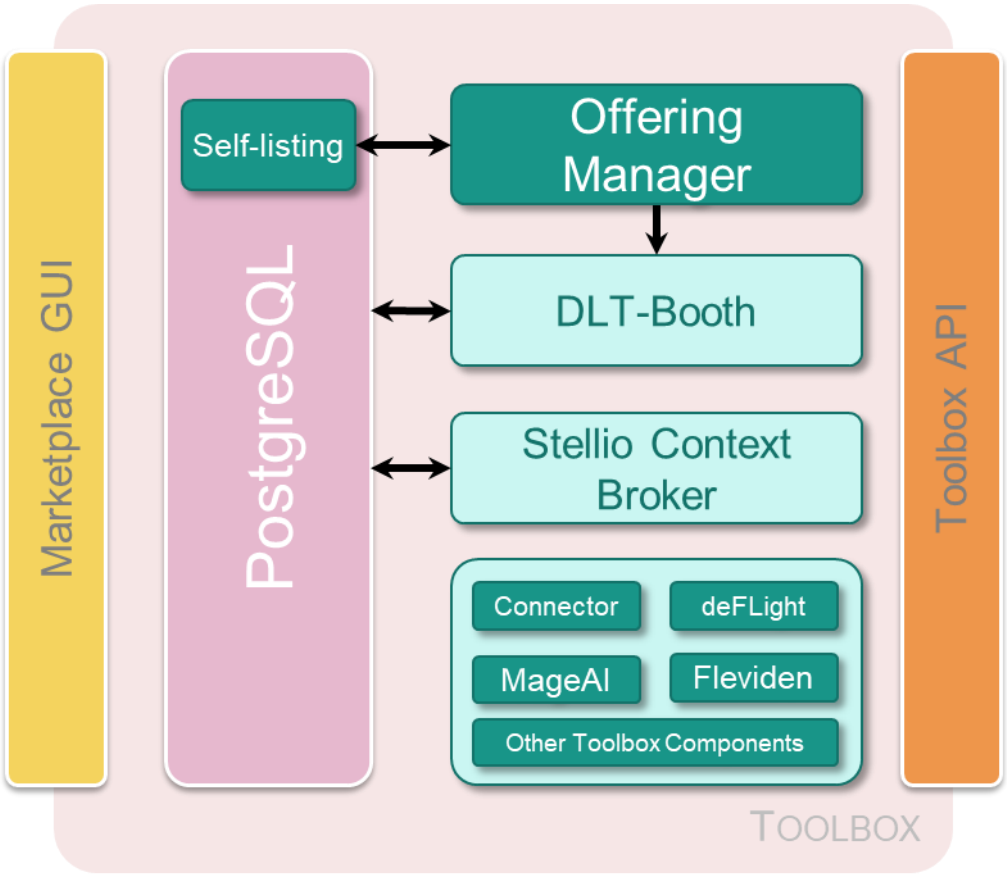


Figure 48: SEDIMARK Toolbox components and storage

Offerings are also stored in the Catalogue, a triple-store database based on the Jena TDB store and Fuseki server. The management of Offerings at the Catalogue is done through a custom handler for encapsulating Offering within named graphs, which will allow the creation and removal of Offerings without leaving the possibility of any orphan RDF nodes which can be caused if Offerings are managed within one default graph.

For the decentralised Catalogue, a Catalogue Coordinator within the domain of a Participant hosting a Catalogue, uses the management custom handler for distributing the Offerings retrieved from Self-Listings. In the case of discovery, Consumer SPARQL requests are modified so that Federated Queries are applied.

Over time the triple store can become fragmented and grow inefficiently, which normally occurs when CRUD operations are applied to the RDF graphs in the triple store. The compaction function provided by the Fuseki API addresses this by creating a new compacted version, copying over the current state of the RDF graphs into the new store, and switching to it once the process is done. The process can be done while the Catalogue is running and therefore does not affect availability.

7 Conclusions

In conclusion, this deliverable has presented the comprehensive design and initial implementation of the foundational technical enablers for the SEDIMARK platform. The work detailed herein successfully establishes the critical pillars of interoperability, artificial intelligence, distributed ledger technology, and storage, which are essential to realise the project's overarching vision for a secure, trusted, and efficient marketplace for data and services. These components collectively represent a significant advancement from the project's starting point, moving the platform closer to its goal of a high Technology Readiness Level (TRL) demonstration.

A cornerstone of the reported achievements is the establishment of a multi-faceted interoperability framework. By standardizing on NGSI-LD for data assets and developing a bespoke Marketplace Information Model for describing participants, offerings, and assets, a common semantic language has been created for the ecosystem. This is a crucial step toward ensuring that data can be discovered, accessed, and reused seamlessly, in alignment with FAIR (Findable Accessible Interoperable Reusable) principles. The development of the Interoperability Enabler, with its suite of tools for data formatting, curation, quality annotation, and validation, provides the practical mechanisms to enforce these standards and transform heterogeneous data sources into compliant, high-quality assets ready for exchange. The introduction of the Offering concept, which allows multiple assets to be bundled, represents a key innovation that provides greater flexibility for data providers compared to existing models.

In the domain of artificial intelligence, the AI Enabler introduces a sophisticated and versatile suite of tools designed to support both local and distributed machine learning scenarios. The provision of advanced models like CrossFormer for multivariate time-series forecasting, along with novel optimization techniques such as structured and unstructured pruning, empowers users to create efficient, high-performance models suitable for deployment in resource-constrained or federated environments. Furthermore, the development of two complementary distributed learning frameworks, the dynamic and fully decentralized deFLight and the extensible Fleviden tool, provides the necessary flexibility to support diverse collaborative training arrangements across multiple organizations and regulatory settings. The innovative, LLM-powered Offering Generator significantly lowers the technical barrier for participation by automating the creation of semantically rich, standards-compliant marketplace offerings from simple metadata descriptions.

These advanced data and AI capabilities are built upon a secure and scalable infrastructure. The private DLT instance, leveraging IOTA Tangle and Smart Contracts, forms the trusted backbone of the marketplace, providing an immutable and non-repudiable ledger for managing participant identities, offering metadata, and facilitating asset trading. This directly addresses the project's core requirement for a system that is secure and trustworthy by design. Complementing this, the distributed Storage Enabler, which utilizes Minio for AI models and NGSI-LD brokers for data, ensures that the heterogeneous assets within the marketplace can be stored, managed, and accessed in a scalable, fault-tolerant, and performant manner.

Looking forward, the components detailed in this document are now primed for the next phase of the project, which will focus on their integration into a cohesive platform and validation within the project's real-world scenarios.

8 Bibliographie

[1] E. I. CIM, «ETSI GS CIM006 - 3NGSI-LD Information model",» ETSI, 2023.

[2] W. Li, G. Tropea, A. Abid, A. Detti et F. Le Gall, «Review of Standard Ontologies for the Web of Things,» chez *Global Internet of Things Summit (GIoTS)*, Aarhus, 2019.

[3] SEDIMARK, «D2.3 - SEDIMARK Architecture and Interfaces. Final version.».

[4] R. Iannella et S. Villata, «ODRL Information Model 2.2,» 15 February 2018. [En ligne]. Available: https://www.w3.org/TR/odrl-model/.

[5] R. Albertoni, D. Browning, S. Cox, A. González Beltrán, A. Perego et P. Winstanley, «https://www.w3.org/TR/vocab-dcat-2/,» 4 Ferbruary 2020. [En ligne]. Available: https://www.w3.org/TR/vocab-dcat-2/.

[6] D. Brickley et L. Miller, «Friend Of A Friend - version 0.99,» 14 June 2014. [En ligne]. Available: http://xmlns.com/foaf/0.1/.

[7] DCMI Usage Board, «DCMI Metadata Terms,» 20 January 2020. [En ligne]. Available: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/.

[8] Internatonal Data Spaces Association, «International Data Space Protocol - Working Draft,» 2023. [En ligne]. Available: https://docs.internationaldataspaces.org/ids-knowledgebase/v/dataspace-protocol/.

[9] Stanford University, «Protégé,» [En ligne]. Available: https://protege.stanford.edu/.

[10] G.-X. E. A. f. D. a. Cloud, «Gaia-X Specifications,» [En ligne]. Available: https://docs.gaia-x.eu/#/.

[11] I. D. S. (IDS), «IDS Knowledge Base,» [En ligne]. Available: https://docs.internationaldataspaces.org/ids-knowledgebase.

[12] W3C, «SKOS Simple Knowledge Organization System Reference,» 18 August 2009. [En ligne]. Available: https://www.w3.org/TR/skos-reference/.

[13] TESK, «Tesk Documentation,» [En ligne]. Available: https://tesk.readthedocs.io/en/latest/.

[14] S. p. (. 101070074), «D3.2 Energy efficient AI-based toolset for improving data quality. Final version,» 2025.

[15] V. F. J. G. David Salinas, «DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks,» 22 February 2019.

[16] Microsoft Corporation, «LightGBM Documentation,» [En ligne]. Available: https://lightgbm.readthedocs.io/en/stable/.

[17] Yandex, «CatBoost is a high-performance open source library for gradient boosting on decision trees,» [En ligne]. Available: https://catboost.ai/.

[18] xgboost developpers, «XGBoost Documentation,» [En ligne]. Available: https://xgboost.readthedocs.io/en/stable/.

[19] H. W. O. M. S. D. S. L. Ria Doshi, «CrossFormer: Scaling Cross-Embodied Learning for Manipulation, Navigation, Locomotion, and Aviation,» [En ligne]. Available: https://crossformer-model.github.io/.

[20] «MLflow 3,» [En ligne]. Available: https://mlflow.org/docs/latest/genai/mlflow-3/.

[21] SEDIMARK, «D4.1 - Decentralized infrastructure and access management. First version,» 2023.

[22] SEDIMARK, «D4.2 - Decentralized infrastructure and access management,» 2025.

[23] SEDIMARK, «D3.3 - Enabling tools for data interoperability, distributed data storage and training distributed AI models. First version,» 2024.

[24] N. A. M. P. M. Ian F. Adams, «Enabling near-data processing in distributed object storage systems,» vol. Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems, 2021.

[25] n. (. A. F. Lead author (surname, «Project name, deliverable number and title (i.e. WITDOM. D2.2 - Functional analysis and use cases identification),» Year (i.e. 2015).

[26] Y. Z. P. Y. H. J. L. L. Buwen Wu, «SemStore: A Semantic-Preserving Distributed RDF Triple Store,» vol. CIKM '14: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge, 2014.

[27] J. D. S. G. W. C. H. D. A. W. M. B. T. C. A. F. a. R. E. G. Fay Chang, «Bigtable: A Distributed Storage System for Structured Data,» Vols. %1 sur %2ACM Trans. Comput. Syst. 26, 2, Article 4 , 2008.

[28] SEDIMARK, «D2.1 - Use Cases Definition and Initial Requirement Analysis».

[29] Institute of Electrical and Electronics Engineers, «IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries,» New York, NY, 1990.

[30] W. W. C. a. S. E. F. P. Ravikumar, «A secure protocol for computing string distance metrics,,» chez *Fourth IEEE International Conference on Data Mining* , Brigthon, 2004.

[31] M. F. D. G. S. J. M. S. a. J. T. R. Egert, «Privately computing set-union and set-intersection cardinality via bloom lters,» chez *Information Security and Privacy*, E. F. a. D. Stebila, Éd., Springer International Publishing, 2015, pp. 413-430.

[32] V. I. B. K. A. M. H. B. M. S. P. D. R. A. S. a. K. S. K. Bonawitz, «Practical secure aggregation for privacy-preserving machine learning,» chez *ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '17*, New York, NY, 2017.

[33] D. J. B. a. T. T. a. A. M. a. X. Q. a. J. F.-M. a. Y. G. a. L. S. a. K. H. L. a. T. P. a. P. P. B. d. G. a. N. D. Lane, *Flower: A friendly federated learning research framework.",* arXiv 2007.14390, 2022, p. v.

9 Annexes

9.1 SHACL Shapes for Offering Validation

```
@prefix dash: <http://datashapes.org/dash#> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix schema: <http://schema.org/> .

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix dcat: <http://www.w3.org/ns/dcat#> .

@prefix sedimark: <https://w3id.org/sedimark/ontology#> .


# Shape to ensure at least one instance of Offering class exists

sedimark:OfferingExistsShape

  a sh:NodeShape ;

  sh:targetClass sedimark:Offering ;

  sh:sparql [

    a sh:SPARQLConstraint ;

    sh:message "At least one instance of sedimark:Offering must exist." ;

    sh:select """

    PREFIX sedimark: <https://w3id.org/sedimark/ontology#>

      SELECT $this

      WHERE {

        FILTER NOT EXISTS {

          ?offering a sedimark:Offering .

        }

      }

    """ ;

  ] .
```

```
# Shape for validating Offering instances
sedimark:OfferingShape
    a sh:NodeShape ;
    sh:targetClass sedimark:Offering ;
    # Offering must have at least one Asset
    sh:property [
        sh:path sedimark:hasAsset ;
        sh:class sedimark:Asset ;
        sh:minCount 1 ;
        sh:message "Each Offering must have at least one Asset." ;
    ] ;
    # Offering must have at least one OfferingContract
    sh:property [
        sh:path sedimark:hasOfferingContract ;
        sh:class sedimark:OfferingContract ;
        sh:minCount 1 ;
        sh:message "Each Offering must have at least one OfferingContract." ;
    ] ;
    # Required properties
    sh:property [
        sh:path dcat:title ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:message "Each Offering must have at least one dcat:title." ;
    ] ;
    sh:property [
        sh:path dcat:description ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
```

```
    sh:message "Each Offering must have at least one dcat:description." ;
  ] ;
  sh:property [
    sh:path dcat:keyword ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:message "Each Offering must have at least one dcat:keyword." ;
  ] .

# Shape for validating Asset instances
sedimark:AssetShape
  a sh:NodeShape ;
  sh:targetClass sedimark:Asset ;
  # Required properties
  sh:property [
    sh:path dcat:title ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:message "Each Asset must have at least one dcat:title." ;
  ] ;
  sh:property [
    sh:path dcat:description ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:message "Each Asset must have at least one dcat:description." ;
  ] ;
  sh:property [
    sh:path dcat:keyword ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
```

```
    sh:message "Each Asset must have at least one dcat:keyword." ;
  ] .

# Shape for validating OfferingContract instances
sedimark:OfferingContractShape
  a sh:NodeShape ;
  sh:targetClass sedimark:OfferingContract ;
  # Required properties
  sh:property [
    sh:path dcat:title ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:message "Each OfferingContract must have at least one dcat:title." ;
  ] ;
  sh:property [
    sh:path dcat:description ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:message "Each OfferingContract must have at least one dcat:description." ;
  ] ;
  sh:property [
    sh:path dcat:keyword ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:message "Each OfferingContract must have at least one dcat:keyword." ;
  ] .
```

9.2 Evaluation Metrics for CrossFormer

In this section, each function in evaluation metrics is provided as below. Before discussing each equation, the notation is defined as: y_i is the ground truth at time i, $\hat{y}_i$ is the predict, the symbol $\bar{}$ means the average, N is the total number of data points, ε means a very small positive constant to avoid division by zero, and s_i is the scaled factor.

MAE (Mean Absolute Error) is a common loss function and evaluation metric used in regression tasks. It measures the average magnitude of the errors between predicted values and actual values, without considering their direction (i.e., it treats all errors equally, whether positive or negative).

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

MSE (Mean Squared Error) is another widely used loss function and evaluation metric in regression problems. It measures the average of the squares of the differences between predicted and actual values.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

RMSE (Root Mean Squared Error) is the square root of the Mean Squared Error. It retains the advantages of MSE (e.g., sensitivity to large errors) while having the same unit as the target variable, making it easier to interpret.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

MAPE (Mean Absolute Percentage Error) is a commonly used metric for evaluating regression models, especially in forecasting. It expresses prediction accuracy as a percentage, showing the average relative error between predicted and actual values.

$$\mathrm{MAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i + \varepsilon}\right|$$

MSPE (Mean Squared Percentage Error) is a regression evaluation metric that calculates the **mean of squared percentage errors** between predictions and actual values. It's similar in spirit to MAPE but squares the percentage error, making it more sensitive to large relative errors.

$$\mathrm{MSPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{y_i + \varepsilon}\right)^2$$

RSE (Relative Squared Error) is a regression metric that measures how well a model's predictions approximate the actual data **relative to a baseline model**, typically the mean of the target values. It helps assess how much better (or worse) a model performs compared to a naïve predictor.

$$\mathrm{RSE} = \frac{\sqrt{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

Considering the MSE may impact by data in diverse value range, the scaled or normalized operation is considered as:

$$\mathrm{ScaledMSE} = \frac{1}{N}\sum_{i=1}^{N}\frac{(y_i - \hat{y}_i)^2}{s_i^2 + \varepsilon}$$

$$\mathrm{NormalizedMSE} = \frac{1}{N}\sum_{i=1}^{N}\frac{(y_i - \hat{y}_i)^2}{(r_i^2 + \varepsilon)}$$

Scaled Log-Cosh is a smooth regression loss function that behaves similarly to Mean Squared Error (MSE) near zero but is **less sensitive to outliers**—like Mean Absolute Error (MAE)—due to its logarithmic growth at large errors. The "scaled" version introduces a scaling factor to control the sharpness of the penalty.

$$\mathrm{ScaledLogCosh} = \frac{1}{N}\sum_{i=1}^{N}\frac{\log(\cosh(y_i - \hat{y}_i))}{s_i + \varepsilon}$$