Call: HORIZON-HLTH-2022-TOOL-11
Topic: HORIZON-HLTH-2022-TOOL-11-02
Funding Scheme: HORIZON Research and Innovation Actions (RIA)

Grant Agreement no: 101095435

Deliverable No. D4.1

Federated Cloud-Based Data Repository

Contractual Submission Date: 31/07/2025

Actual Submission Date: 31/07/2025

Responsible partner: Maastricht University

Funded by
the European Union

Grant agreement no.	101095435
Project full title	REALM - Real-world-data Enabled Assessment for heaLth regulatory decision-Making

Deliverable number	**D4.1**
Deliverable title	**Federated Cloud-Based Data Repository**
Type[1]	R – Report
Dissemination level[2]	PU – Public
Work package number	4
Work package leader	P1-UM
Author(s)	Aris Bonanos (EXUS), Chang Sun (UM), Frederic Jung (VITO), Giorgos Nomikos (EXUS), Pedro Lopez (EXUS), Nam Doan (VITO), Gabrijela Radic (EURICE)
Keywords	Federated Data, FAIR, OMOP, EHR, Data Repository

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HaDEA).

Neither the European Union nor the granting authority can be held responsible for them.

Document History

Version	Date	Description
V0.1	15/01/2025	ToC created
V0.2	11/04/2025	VITO Revised ToC, standardization, OMOP Strategy
V0.3	12/06/2025	Collect inputs from partners
V0.4	10/07/2025	Final draft ready for internal review
V1.0	31/07/2025	Final submission

[1] **Type**: Use one of the following codes (in consistence with the Description of the Action):
R: Document, report (excluding the periodic and final reports)
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.

[2] **Dissemination level**: Use one of the following codes (in consistence with the Description of the Action)
PU: Public, fully open, e.g. web
SEN: Sensitive, limited under conditions of the Grant Agreement

Table of Contents

List of Abbreviations and definitions

AI - Artificial Intelligence
CDM – Common Data Model
COPD - Chronic Obstructive Pulmonary Disease
CT Scan - Computerized Tomography scan
DAA – Data access agreement
DARWIN EU – Data Analysis and Real World Interrogation Network
DDL – Data Definition Language
DICOM - Digital Imaging and Communications in Medicine
DMP - Data Management Plan
EHDEN – European Health Data Evidence Network
EHDS – European Health Data Space
EHR - Electronic Health Record
ETL – Extract Transform and Load
FAIR - Finable, Accessible, Interoperable, Reusable
FHIR - Fast Healthcare Interoperability Resources
GDPR - EU General Data Protection Regulation
HITL – Human In The Loop
JVM – Java Virtual Machine
LLM - Large Language Model
LTS - Long term support
NER - Name Entity Recognition
OHDSI – Observational Health Data Sciences and Informatics
OMOP - Observational Medical Outcomes Partnership
RAG - Retrieval-Augmented Generation
REALM - Real-world-data Enabled Assessment for heaLth regulatory decision-Making
RIANA - REALM Intelligent Analytics Dashboard
RWD – Real-World Data
VCF – Variant Call Format

Executive Summary

Deliverable D4.1 documents the development and deployment of the federated, cloud-based data repository infrastructure designed under Task 4.1. REALM data repository is a cornerstone of the REALM platform, enabling secure, privacy-preserving, and interoperable access to anonymized, multimodal real-world data (RWD) and synthetic data to support the evaluation of AI-based medical device software (MDSW). The architecture follows a federated approach that accommodates direct access to data. This access mode allows national nodes to host and manage data locally while enabling AI model evaluation within their secure environments.

The repository is powered by a robust infrastructure stack integrating Docker, Kubernetes, Kafka, PostgreSQL, and OHDSI tooling. These technologies collectively support data ingestion, transformation (extract, transform, load (ETL) operations), cohort generation, and AI evaluation through the RIANA dashboard, and semantic alignment using the OMOP Common Data Model. Extensions were developed to support genomics, radiology, and oncology use cases, and synthetic data generators were implemented for scenarios where real-world data is limited or incomplete. Deployment requirements and configurations are documented to facilitate the setup of the national REALM data nodes.

A central component of this infrastructure is the standardized data catalogue (T4.3) based on the OMOP Common Data Model, extended to support genomics (G-CDM), radiology (R-CDM), and oncology domains. A comprehensive ETL pipeline and mapping methodology, combining OHDSI tools and custom scaffolding, was developed to enable the transformation of heterogeneous data into OMOP Common Data Model compliant datasets, with real-world and synthetic datasets used to validate the pipelines.

The work is further supported by extensive engagement with data providers, resulting in a cooperation model that emphasizes shared value over direct compensation. This includes technical support, benchmarking insights, and opportunities for joint dissemination.

Looking forward, this deliverable opens several promising directions for future exploration and uptake. These include operationalizing the indirect access model in alignment with national and EU-level regulatory frameworks, such as those defined under the European Health Data Space (EHDS). REALM's national nodes could potentially serve as complementary infrastructures under the governance of Health Data Access Bodies (HDABs). Further opportunities lie in expanding the network of participating data providers, improving data and model interoperability with EHDS standards, and strengthening automated tooling for data mapping and quality assurance.

1 Introduction

Deliverable 4.1 is part of REALM Work Package 4 (Task 4.1), focusing on the development of a repository to host federated, secure, and interoperable real-world data (RWD) and synthetic data and to host the models for adversarial post-market evaluations and synthetic data generations. This deliverable contributes to the overarching objectives of REALM by implementing a secure, FAIR-compliant, and standardized data architecture that facilitates the evaluation of AI models for healthcare applications. By providing a data repository for the evaluation of AI-driven Medical Device Software (MDSW), developers can assess the performance of their models against real-world data and/or synthetic data from multiple data sources, whereas regulators can obtain evaluation process and results on the model performance.

This task focused on design and implement a cloud-based federated repository for collecting, standardizing, and managing multimodal health data. The data originates from heterogeneous sources

including electronic health records (EHRs), laboratory reports, medical sensor data, and medical images. It is harmonized using the OMOP Common Data Model (CDM) to enable structured storage and interoperability. In addition to RWD and data, the repository also host internal models for synthetic data generation as well as post-market evaluation of the synthetically generated data using MinIO. More details were covered in the previous submitted Deliverables 3.1

This deliverable summarizes the key activities undertaken to implement Task 4.1:

- Design of the federated data architecture, including decisions regarding tooling, interoperability, and infrastructure;
- Development of the REALM standardized data catalogue, based on OMOP and extended for genomics and imaging domains (G-CDM and R-CDM);
- Implementation of ETL pipelines to support the transformation of diverse data sources into OMOP-compatible datasets, including structural and semantic mapping strategies;
- Integration of intelligence modules to support feature extraction, cohort building, and model application using OHDSI tools and AI components;
- Demonstration using synthetic and real-world datasets, including examples like Synthea and the Kaggle COPD dataset;
- Governance and sustainability planning, with emphasis on stakeholder-driven design, data quality, privacy protection, and knowledge management.

The structure of this deliverable is as follows. Section 2 describes the federated architecture and design choices and the infrastructure development process. Section 3 presents the standardized OMOP-based data catalogue and associated tools. Section 4 discusses the data sources and outlines the ETL strategy for transforming raw datasets into OMOP-compatible formats. Section 5 and Section 6 conclude the work and deliverable and present potential future work.

2 The Federated Cloud-Based data repository

A central theme of the REALM platform is enabling federated access to sensitive medical data. Data federation is a technique to solve the problem of managing data scattered across multiple systems and databases. In REALM, we set up several federated data nodes where we collect and host different datasets and facilitate users to access and query data from multiple sources (Figure 1). Within a country a national REALM node was established to host data, with dedicated data access agreements governing the transfer and use of this data. Below we will outline REALM's approach and technical solution to data federation.

2.1 The Federated Realm Architecture

REALM's federated architecture follows a comprehensive analysis of stakeholder needs carried out in the project and presented in D4.4, while the full architecture was given in D3.1. The needs are organized in three categories as: 1) user requirements, 2) security & privacy requirements, and 3) ethical, privacy, and data protection requirements. The federated architecture has been designed to comply with these requirements. The use of the OMOP standardized data model ensures that aspects such as data annotation through labelling and tagging, metadata and interoperability are built into the system. Further details are elaborated in Section 3. In this section, we report on the implemented federated cloud data repository through the direct access mode (Section 2.1.1) and the envisioned extensions to this architecture through indirect access mode (Section 2.1.2).

A critical aspect to both the direct and indirect access modes concern the model itself. From a model developer perspective, REALM's value proposition is that it enables them to evaluate their model against private data. To achieve this respecting the data federation aspects requires that the model must be transferred to the data. However, a model developer might be hesitant to provide 3rd party access to their model. REALM's solution to this is for the model developers to only provide (temporary) access to the model location (e.g. developer's repository) and this location is then supplied to a Kubeflow component that loads the model. Once the Kubeflow pipeline is executed, all artefacts associated with it are deleted, including the model's artefact. Alternatively, the model developer can Dockerize their component and provide the docker location. Similarly, a docker loader component is executed during the pipeline operation and at the conclusion, all artefacts are deleted.

2.1.1 Direct Access

The deliverable 3.1 "The Federated REALM Architecture" outlines the project's software architecture, as it was co-created to address user needs. For completeness, here we summarize the architectural design choices associated with the data repository reported in the present deliverable. The REALM platform adopts a federated architectural design to meet the stringent security and privacy requirements associated with processing real-world data (RWD) from the medical domain.

Due to legal, ethical, and institutional constraints, access to restricted RWD is typically only granted after signing a data access agreement between the data source and the party requesting access to the data, such as a model owner wanting to evaluate its model. This process is slow and cumbersome, and it does not scale well when a party needs access to multiple data sources, which is an essential requirement for thorough model evaluations.

To address this, REALM establishes national "nodes" which are authorized to ingest RWD under data access agreements (DAA) and for clearly defined purposes. These nodes are deployed in multiple countries in a federated manner to avoid the need for cross-border data transfers. If a national node is unavailable and sharing data with a neighbouring country's node is permitted, using that node is of course also possible. The nodes do not further disseminate the data, instead, they act as secure processing environments within the federated platform where multiple data sources are available in the form of a 'data catalogue'. Within each node, AI model evaluations can be executed directly against the local RWD within a protected environment. These evaluations are standardized and automated, and model owners never get access to the underlying evaluation data. Only evaluation results based on performance metrics are made available to them.

Secure communication between nodes: To support this secure and automated model evaluations within REALM nodes, several interoperable software service components have been integrated. The secure communication to REALM nodes relies on the middleware API, which enables structured information exchange between REALM nodes. The middleware is built upon Apache Kafka, a widely adopted distributed event-streaming platform. Kafka was selected for its proven performance in handling high-throughput, fault-tolerant, and asynchronous communication. These are critical features for federated settings where nodes may vary in availability and responsiveness. Kafka ensures that messages, such as evaluation requests and results, are reliably queued and delivered, even in cross-border or intermittent connection scenarios.

Evaluation execution: Then, each evaluation request consists of a structured set of instructions packaged in a YAML file. YAML is a lightweight, human-readable format commonly used for configuration and workflow orchestration. The instructions in the YAML file are executed locally within the node by the AI Orchestrator, which is built using Kubeflow, an open-source machine learning platform optimized for Kubernetes. Kubeflow was chosen for its ability to containerize, scale, and monitor AI workflows consistently across heterogeneous environments. It allows nodes to manage

complex, multi-step evaluations as repeatable pipelines, ensuring reproducibility and isolation of every evaluation run.

Return output: Once an evaluation is completed, the results, which are limited to performance metrics and model metadata, are compiled into a standardized Model Card. This card serves as a transparent and interpretable summary of the model's behaviour on the specific dataset. It is then returned to the originating node via the middleware API, where it can be reviewed by the model owner without ever exposing the underlying patient-level data. To achieve this, several accompanying software services components must be in place. The REALM nodes securely exchange information via the middleware API, based on Kafka. The information exchanged consists of the evaluation instructions for a particular MDSW application, packaged in a YAML file. The evaluation results are reported on the model card, which is also communicated back to the requesting node via the middleware API.

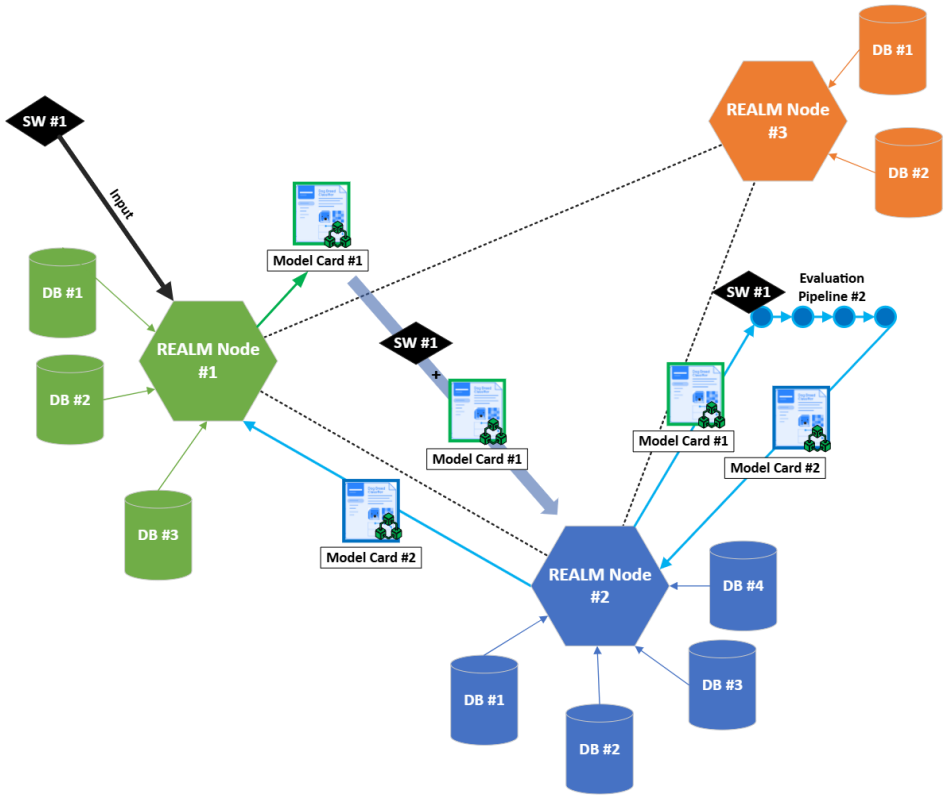


Figure 1. An overview of REALM federated nodes

An overview of REALM federated evaluation mechanism is illustrated in Figure 1. The diagram depicts a scenario where an AI model is evaluated on RWD hosted in a different node than the one from which the evaluation request originates. Specifically, a model evaluation claim is created through RIANA dashboard in REALM node #1. This claim is encapsulated and formatted as a YAML file and is sent via the middleware API to REALM node #2, there the target data resides. The evaluation is executed end-to-end within node #2, ensuring data sovereignty. Only the evaluation results, such as performance metrics and model metadata, are returned to node #1 and appended to the corresponding Model Card.

2.1.2 Indirect Access

In the REALM project, data sovereignty and provider autonomy are central concerns. We have conceptualised an indirect access strategy that allows data pools to participate in REALM without granting direct access to their sensitive data, as done in the direct access implementation shown in Figure 1. The direct access mode assumes that the data can be integrated into the national nodes, however, in some cases, due to regulatory, institutional, or technical constraints, data provides are

unwilling to or unable to transfer RWD to the national nodes. To accommodate such cases, REALM has conceptualized an indirect access strategy, which allows data providers to perform model evaluation without relinquishing any control over their data.

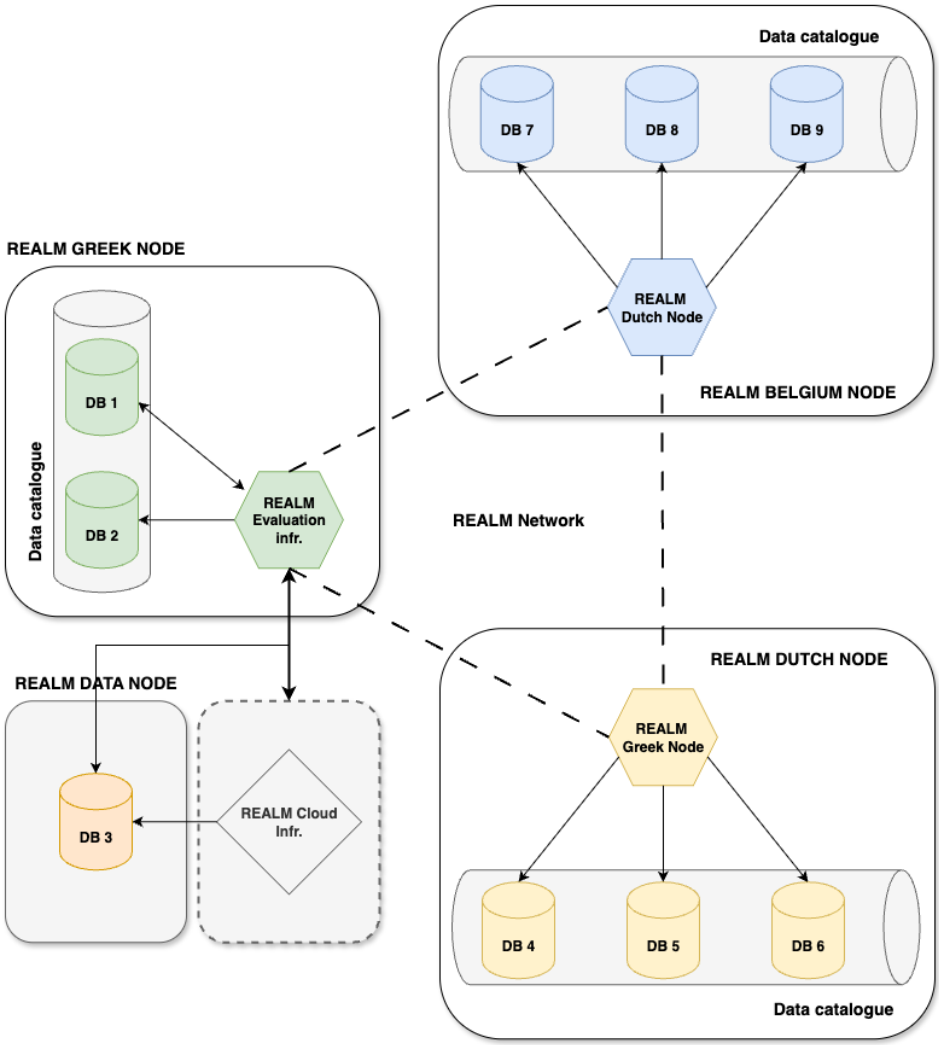


Figure 2 Envisioned indirect node configuration in REALM federated architecture

Under the indirect access model, the data provider operates as a REALM Private Node, hosting their data locally and executing AI evaluation workflow on-site. In this regard, we bring the REALM evaluation tools and pipeline and model to be evaluated at the data provider's site. This includes the complete AI evaluation stack, such as the cohort generation, model execution and reporting modules. This is achieved by providing the evaluation instructions and access to the model from one node to another by sharing only the YAML file between nodes. In this regard, the private node is responsible for (1) executing on their own infrastructure the evaluation workflow on local data (2) ensuring that data are secured in their system and never leaves their premises and (3) returning only the results of the evaluation that consist of a model card (described in Deliverable D3.3) that contains performance metrics and high-level demographics and cohort statistics. This approach is more like how the DARWIN EU network operates

The primary limitation of the indirect access mode lies in the requirement for the entire REALM evaluation framework to be deployed and executed within the data providers' infrastructure. This entails the installation and maintenance of multiple interconnected software components, as well as

ensuring access to sufficient computational resources capable of executing demanding AI workflows. For many institutions, particularly those with limited IT capacity, this represents a significant technical and operational burden.

In alignment with the scope of the REALM minimum viable product, only the direct access mode is released and supported to date. This decision primarily reflects the current limitations in technical feasibility, maintainability, and broad stakeholder participation. However, the indirect access mode is discussed as a potential future implementation that makes the REALM platform more attractive and flexible to accommodate data owners' needs.

2.2 Legal Support

To enable secure and lawful access to external real-world data sources in support of the federated REALM data repository, the consortium established a legal and governance framework to support data nodes establishment. Following a General Assembly vote held in June 2025, the consortium mandated Maastricht University (UM) and VITO to act on its behalf in negotiating and signing Data Access Agreements (DAAs) with third-party data providers. This mandate was formalized through a Letter of Mandate, signed by all partners, which clearly authorizes UM and VITO to conclude DAAs using a non-negotiable, consortium-approved template. The process was supported by significant contributions from the legal departments of both UM and VITO, ensuring full alignment with applicable regulatory and data protection requirements. The template DAA ensures compliance with the GDPR, detailing responsibilities related to data pseudonymization, security measures, purpose limitation, and data subject rights. It defines both parties as independent controllers and embeds safeguards for confidentiality, breach notification, and data destruction protocols. As of this deliverable, the mandate framework and template DAA have been finalized and disseminated. The next steps include identifying and engaging with external data holders, operationalizing data transfer in line with the agreed terms, and ensuring that local REALM nodes implement technical and organizational safeguards as per the DAA and project-wide compliance protocols.

2.3 Cooperation and benefits

Throughout the REALM project, we engaged in numerous discussions with data owners to define mutually beneficial data access agreements. A recurring and essential topic in these discussions was the structure of compensation and cooperation. To foster trust, transparency, and sustainable collaboration, we developed a flexible set of compensation and engagement options tailored to the needs and capacities of each partner.

A fair compensation package has been designed to cover the data providers effort in making the data available. Setting up secure access to data can represent a significant investment for data owners, not only in terms of potential hardware requirements, but also because it demands time and expertise from IT staff and engineers. Thus, the compensation facilitates the time and effort for the investment of the data owner for bridging the data and ensure security. The compensation options, summarized in Table 1, include technical support, shared evaluation outcomes, involvement in strategic initiatives like the REALM Academy, and insights into the real-world usability of their data.

While designing our cooperation model, we deliberately limited the emphasis on direct financial compensation such as one-time payment and/or per-evaluation payments (a fixed amount per AI model evaluation using the provider's data). These options of compensation can be easily implemented but we found they were not always aligned with the collaborative spirit of REALM. The project's core objective is to build a trusted ecosystem for evaluating AI in healthcare, rooted in shared

values. Thus, we prioritized alternative forms of support, such as capacity-building, co-authorship, and shared tools, over purely transactional relationships.

Table 1:Compensation & Value Package for Data Partners - range of compensations, services, and collaborative benefits offered to data providers participating in the REALM project

Compensation Type	Description
Set-up of OHDSI Tooling	Installation and configuration of OHDSI stack (e.g., Atlas, WebAPI, OMOP CDM).
Access to AI Evaluation	Free or prioritized access to run/ submit or observe AI model evaluations.
Trained Synthetic Data Generator	Delivery of a generator trained on their data to create privacy-preserving synthetic data.
Data Maintenance Funding / Service	Financial or technical support for internal efforts like data cleaning, de-identification, and OMOP mapping.
Early Access to Results	Priority access to insights and performance metrics of AI models tested on their data.
REALM Academy & Use-Case Events	Participation in training, demonstrators, and strategic events as a key contributor.
Data quality / usability	Feedback on the real-world usability of their data, identify strengths and gaps in data readiness for medical AI.
Assistance in REALM node setup and maintenance.	REALM will provide assistance to the data provider in setting up a secured transfer tunnel and/or setting up their REALM private node.

2.4 The Infrastructure Development

During the preparation of the REALM proposal, the consortium had identified MedOMICS [https://www.medomics.ai/] as the framework for the federated cloud-based data repository catalogue, given that -according to the literature- it consisted of five modules that would be needed in the context of REALM as well, those being:

i) Input: for integrating and validating diverse patient databases into a common framework using data representation and ontologies;
ii) Extraction: for state-of-the-art quantitative image and natural language processing (NLP) analyses that are performed from the database to extract features;
iii) Discovery: for data exploration to uncover new associations between different types of features;
iv) Learning: for integrating multiomics data using AI/machine learning techniques to generate prediction models for the relevant clinical use-case; and
v) Application: for developing prediction models and decision support systems (DSSs) for unmet clinical needs

The consortium initially considered the medOMICS solution but identified several obstacles in its implementation. The medOMICS GitHub repository [https://github.com/medomics/] seemed inactive, with the last update at the time (end of 2023) being from January 2022. Additionally, no code relating to the medOMICS infrastructure was available. Through the REALM consortium, some medOMICS contributors were identified and contacted to explore the potential for collaboration and revival of the medOMICS effort within REALM. The subsequent discussions concluded that medOMICS in fact consisted of the "medOMICS labs" and the "medOMICS infrastructure", with the former being currently pursued by the developers but the latter being of interest for REALM.

Given this situation and the time pressure to commence developments, an alternative approach was pursued. We investigated the suitability of CODA [https://www.coda-platform.com/] but discovered that this is not natively compatible with the OMOP common data model. Workarounds to this were identified, for example through the OMOPonFHIR transformation utility [http://omoponfhir.org/]. Our main concern was that both CODA and OMOPonFHIR are rapidly evolving with frequent new releases

and minimal documentation, so using these was considered risky as their functionality and suitability had not been completely tested.

Our alternative approach was to develop the federated architecture scheme described in Sec. 2.1 and to incorporate the OHDSI tooling (described in Sec. 3) which are incorporated through the RIANA dashboard to achieve the extraction and discovery features. Further, the requirements for the REALM infrastructure became clearer during the development, rendering some aspects of the medOMICS infrastructure irrelevant or covered by other components of the REALM platform. Specifically, regarding the learning and application modules, as mentioned during this deliverable, REALM platform is intended exclusively for the evaluation of AI models within the healthcare domain. These models are provided in a pre-trained state by their respective developers or institutions, and no training or development processes are conducted within the REALM nodes. The platform's role is strictly confined to evaluation activities, ensuring a clear separation between model creation and model evaluation.

2.4.1 Minimum REALM Node Server Requirements

As the REALM architecture evolved and matured, the core set of technical components and supporting technology stack used by each consortium partner were consolidated (See in Deliverable D4.4). This process enabled us to define the minimal technical specifications required for the deployment of a fully functional REALM Node. These components include tools for inter-node communication, orchestration of AI evaluation pipelines, and the deployment of the REALM Intelligent Analytics (RIANA) dashboard.

Given that REALM Nodes are expected to operate multiple services concurrently, the specifications outlined below represent a baseline configuration. Actual requirements may vary depending on the complexity and scale of the AI pipelines executed within the node. For high-throughput or resource-intensive workflows, additional computational resources may be necessary to ensure optimal performance.

Operation system: The recommended environment is Ubuntu LTS. Alternative Linux distributions may be used, but compatibility and stability should be carefully validated.

Storage: A 1TB of storage is recommended to support the installation of all required services. Additional storage may be necessary depending on the volume and nature of AI pipeline executions. Disks are suggested to be of type NVMe or SSD for faster transfer speeds in comparison with mechanical disks.

Processor: The system should include a CPU with at least 8 threads and modern architecture (released in 2022 or later) to accommodate the processing demands of AI models evaluation and orchestration tasks.

Memory: 32 GB of RAM is advised. This should be scaled proportionally if the node is expected to support larger or more complex pipeline executions.

2.4.2 Server Configuration

The first step to set up a REALM node is to install the operating system (Ubuntu LTS is recommended). Ubuntu 24.04.2 (Noble Numbat) is the latest available LTS version during the writing of this deliverable. After installation of the operating system, a series of essential services must be deployed to enable the

execution of the REALM pipelines. An overview of the required services and their core functionality is given in Table 2, while a complete guide for their installation is given in Annex 8.

Figure 3 shows a workflow of the required services. At the foundation, Ubuntu LTS serves as the base operating system. On top of this, Docker is used to containerize all services, enabling modular deployment and easier maintenance. Traefik acts as a reverse proxy and load balancer, managing ingress traffic to the various services. Kafka, supported by Kafka Manager and Kafdrop, provides the distributed messaging backbone for inter-node communication.

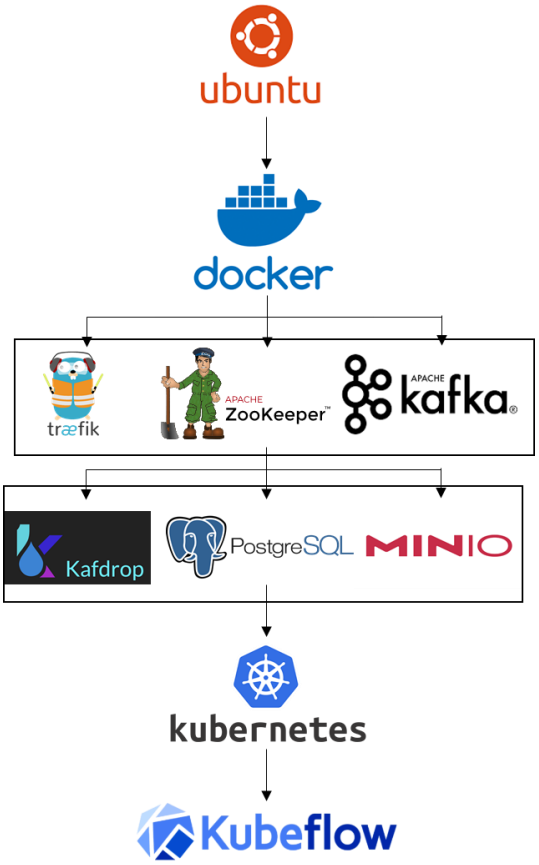


Figure 3 Services required for a REALM node.

The platform uses PostgreSQL as its relational database system for storing metadata and orchestration state, while MinIO provides a secure, S3-compatible object storage layer for model artifacts and pipeline outputs. All services are deployed on Kubernetes, which manages container scheduling, scaling, and monitoring across the node. Finally, Kubeflow orchestrates the AI evaluation workflows through declarative pipelines, ensuring repeatable, modular, and secure execution of complex model evaluation tasks.

Once a server has been configured with the required services, it is fully equipped to operate as a REALM Node. This means that the necessary software infrastructure is in place to support two core functionalities:

1) Medical Device Software (MDSW) developers can submit their applications for evaluation against data residing either within the local node or in another REALM node across the federation;
2) The node can receive and execute the YAML file containing instructions for the evaluation of the AI-driven MDSW that was created in another node.

Table 2 REALM Server essential components and core functionalities

Node Services	Functionality
Ubuntu OS (https://ubuntu.com)	The Operating System (OS) forms the foundational software layer that manages the server hardware and provides essential services for other applications. It is compatible with modern DevOps tools and environments (e.g. Docker and Kubernetes)
Docker (https://www.docker.com/)	Docker enables the packaging of applications and their dependencies into isolated, lightweight containers that can run consistently across different environments, ensuring the same performance across development, testing and production environments. A further advantage to docker is that it scales efficiently and avoids dependency conflicts.
Traefik (https://traefik.io/traefik)	Traefik acts as a dynamic reverse proxy and load balancer that routes incoming HTTPS traffic to the appropriate services within the Docker environment, automatically discovering services through Docker labels, simplifying configuration and deployment. It also handles SSL/TLS certificate management ensuring secure connections and supporting high availability in a containerized infrastructure.
Zookeeper (https://zookeeper.apache.org/)	It ensures that services operate reliably and in coordination, preventing conflicts and maintaining high availability, particularly in platforms where distributed state management is required. It maintains configuration information, naming, synchronization and distributed coordination.
Kafka (https://kafka.apache.org/)	A high-throughput, low-latency distributed messaging system designed for real-time data streaming and processing, enabling decoupled communication between microservices or components. It is particularly suited for large-scale data ingestion, logging and analytics, ensuring message logging and delivery in the correct order.
Kafdrop (https://github.com/obsidiandynamics/kafdrop)	This is a web-based interface for viewing and managing Kafka clusters, enabling topic visibility, partitions, messaging, consumer groups and broker status, thus forming a valuable debugging and monitoring tool.
PostgreSQL (https://www.postgresql.org/)	PostgreSQL is an open source relational database management system, used for storing structured data with consistency, reliability and ACID compliance, making it essential for applications requiring complex queries. It supports advanced data types and is particularly suitable for storing user data.
MinIO (https://min.io/)	MinIO is an S-3 compatible object storage service used to store unstructured data such as files, images and logs. It provides scalable storage that can be deployed on-premise or in containers, making it ideal for microservices and distributed applications. It complements the relational database, since it can handle binary data that don't typically fit well into traditional databases.
Kubernetes (https://kubernetes.io/)	Kubernetes automates the deployment, scaling, management and networking of containerized applications, ensuring high-availability, load balancing, self-healing and rolling updates of services. By abstracting infrastructure complexity, it makes it easier to manage large-scale microservice architectures and enables consistent operations.
Kubeflow (https://www.kubeflow.org/)	Kubeflow is built on kubernetes for managing the full machine learning lifecycle, from experimentation and training to deployment and monitoring. It is essential for platforms including ML components that require reproducibility, scalability and automation, ensuring that workflows are standardized, scalable and can integrate into DevOps/MLOps pipelines.

A further note on the inclusion of a file/object storage component (MinIO) in the node infrastructure. The OMOP standardized data model used throughout the REALM platform is designed for structured clinical and observational data, and therefore images are not natively supported. Support for images can be handled by storing the image-related metadata in the structured database and OMOP format

and also including a reference/link to the raw image location. The images themselves are stored in the object storage which the metadata references.

2.4.3 Installing the RIANA Dashboard

Users interact with the REALM platform through RIANA dashboard, either to submit a MDSW for evaluation or to follow up with the post-market evaluation of an MDSW. The RIANA dashboard has been dockerized, and detailed instructions on how to deploy it are given in Annex 8.3. The deployment of the RIANA dashboard completes the server configuration into a REALM node.

3 The REALM Standardized Data Catalogue

3.1 Towards a Standardized and FAIR-Compliant Data Catalogue for REALM

With the software infrastructure and federated orchestration mechanisms in place, a consistent and interoperable approach to data representation for the REALM Nodes becomes essential. To ensure that AI model evaluations across nodes operate on harmonized data structures, the REALM platform incorporates a standardized data layer. This enables seamless communication between tools, analytics across sites, and compliance with European data governance expectations. The following section introduces the REALM Standardized Data Catalogue, which underpins this data layer.

The REALM project (Task 4.3) has designed a data catalogue that adopts rigorous structural and semantic conventions. From the onset, we emphasized that standardization is crucial: consistent data storage, indexing, and cataloguing are foundational to ensure interoperability, scalability, and long-term impact. To that end, we adopted the OMOP CDM as the backbone of our approach, given its broad community support and alignment with regulatory and research needs. This decision allowed us to adhere to the FAIR principles, ensuring that data is Findable, Accessible, Interoperable, and Reusable. Moreover, the OMOP CDM has also proven its efficiency in federated settings, enabling distributed analysis across diverse data partners enabling cross-border health data collaboration. This methodology also gives more flexibility in defining access type to the federated strategy we are proposing. Moreover, our standardized setup facilitate integration with data partners, enabling a plug-and-play architecture where new contributors can easily align with our ecosystem. As a result of using OMOP CDM, there are many European OMOP data partners available with a constantly growing number of sources that can be easily connected to REALM. This reinforces our strategic alignment with the emerging European initiatives such European Health Data Space (EHDS)[3] or Darwin EU[4].

The vision behind the OMOP REALM data catalogue emerged from the need to create a unified, interoperable framework capable of linking a broad range of heterogeneous data sources across clinical, genomic, and imaging domains (Figure 4). By indexing all datasets within a standardized structure, the catalogue enables coherent reporting, streamlined cohort definition, and supports the use of OMOP extensions such as G-CDM (Genomics) and R-CDM (Table 3) or any other person-centric extension that respect the fundamental OMOP CDM constraints such as person and concept keys (with example given in Figure 9 & Annex 3). These extensions introduce new, compatible tables that enable the catalogue to include references to source-specific files such as DICOM for imaging and VCF for genomics. This design allows patient records in the OMOP data source to be systematically linked back to their original supplementary files when needed. Such traceability is particularly valuable for AI models that require access to source input formats. Finally, this approach allows us to capture diverse

[3] https://www.european-health-data-space.com/
[4] https://www.darwin-eu.org/

data modalities under a single, FAIR-aligned infrastructure, ensuring consistency, traceability, and readiness for AI model evaluation.

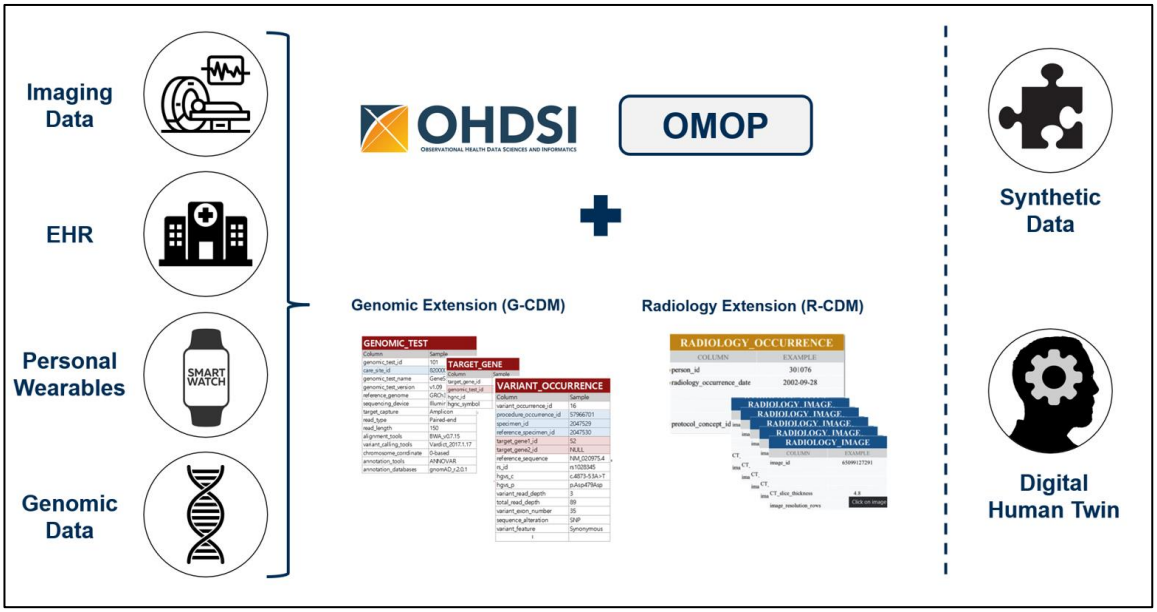


Figure 4: REALM vision of the OMOP Data Catalogue

Table 3: Used OMOP extensions and how REALM meets demonstrator requirements – updated table described in REALM D4.5

Extension Name	Description	Data Type	Publication	Demonstrator
G-CDM	An extension for representing genomic data within the OMOP Common Data Model, facilitating the integration of genetic information into healthcare research.	Genomic Data	(Shin et al., 2019)	PGX2P
R-CDM	Designed to handle DICOM (Digital Imaging and Communications in Medicine) files, which are commonly used in radiology and medical imaging. It allows the integration of imaging data into OMOP-based research.	Medical Imaging Data	[1](Park et al., 2022)	Dune AI,

3.2 OHDSI Tooling

To support the deployment of OMOP-CDM–standardized data infrastructure across REALM national nodes, we have containerized a tailored selection of the OHDSI tools using Docker (listed in Table 4). This enables replicable deployments in heterogeneous environments, reducing the technical burden for node operators. All tools are made available as Docker containers hosted on our private REALM repository. Each container includes a README with step-by-step setup instructions to guide local deployment and configuration. From the set of OHDSI tools available we distinguish (1) the core tools required for REALM workflow and (2) optional tools for data mapping and ETL support. The first category is essential for the REALM federated operations (WebAPI) and for the HITL (Atlas) while the second category is recommended to facilitate transformation of data sources into the OMOP CDM.

Table 4: OHDSI Tooling and their usage in the REALM Project – Key OHDSI tooling used within the REALM project to support data standardization, cohort generation, quality control and the evaluation workflow. Each tool has a specific role in bringing a new data source in the REALM data catalogue and ensure its integration within the REALM architecture.

Tool	Description	Usage in REALM Project
WhiteRabbit (*github.com/OHDSI/WhiteRabbit*)	Scans source data to generate a detailed structured summary of a source dataset table used to guide ETL design: ScanReport.	Used during structural mapping and as input for Rabbit-In-a-hat.
Rabbit-In-a-Hat (*github.com/OHDSI/WhiteRabbit*)	Graphical tool for structural mapping mappings and based on ScanReport.	Used during structural mapping. Drives the ETL implementation.
Usagi (*github.com/OHDSI/Usagi*)	Suggest OMOP concept IDs for source codes using term similarity with limited results. Allows manual review and approval.	Can be used to map source vocabularies to OMOP standard concepts.
ATLAS (*github.com/OHDSI/Atlas*)	Web-based interface to design, execute, create highly detailed cohort definitions based on WebAPI.	Used for creating and testing cohort definitions by the HITL (see section 2.7, D3.1)
WebAPI (*github.com/OHDSI/WebAPI*)	RESTful API backend for OHDSI tools. Enables communication between tools like ATLAS and the OMOP CDM databases.	Supports cohort generation, study execution, and integrates with other tools like the RIANA dashboard.
ACHILLES (*github.com/OHDSI/Achilles*)	Collection of analytics packaged in an R library to be perform against OMOP data sources.	Can be used for data exploration, database profiling.
Heracles (*github.com/OHDSI/Heracles*)	(similar as OHDSI Achilles) Collection of analytics embedded within WebAPI allowing the generation domain-specific concept distribution of generated cohort definitions.	Used to report evaluation dataset distributions to be stored in the model card.
Data Quality Dashboard (DQD) (*github.com/OHDSI/ DataQualityDashboard*)	Performs data quality checks against an OMOP database.	Used for data quality assurance after ETL implementation.
ATHENA (*github.com/OHDSI/Athena*)	Online tool to browse and download OMOP standardized vocabularies.	Can be used by data analysts and model provider for manual mapping of source codes/ vocabulary to the OMOP CDM.

In addition to standard OHDSI tooling, we have developed a utility collection called REALM Utils. This suite includes scripts, DDLs, and lightweight ETL scaffolding designed to assist with a broad range of OMOP-related operations. These are also available in the REALM GitHub repository and come with usage documentation.

Key components include:

- OMOP Schema Deployment:
  - DDL and SQL scripts to deploy a new OMOP CDM schema (clinical tables following OMOP CDM v5.4) including R-CDM extension and Patient Supplement extension (See Annex 3).
  - Scripts to create the OMOP vocabulary schema and insert vocabulary data.
  - Constraints and extensions (including OMOP-CDM and derived tables from extensions).
- ETL Scaffolding Framework:
  - Python-based orchestration for running ETL pipelines.
  - SQL template logic to transform source data into OMOP CDM format.
- Analytics Integration:

  - Scripts to run standard OHDSI analytics packages such as Achilles and Heracles for characterization and quality checks.
- Support for HITL Dataset Engineering:
  - Python source code for the Query Builder, enabling reengineering of evaluation datasets from cohort and concept definitions for the HITL.
- Specialized Tools:
  - A DICOM metadata extractor that parses DICOM files and maps key metadata fields to OMOP CDM tables (R-CDM + OMOP clinical tables)
  - A Python script for parsing sample CSV or TSV input files to extract feature names and data types, designed to populate metadata inputs ("Features" section of the model card) into the RIANA Dashboard.

3.3 Development of the Standardized Data Catalogue

The creation of the standardized OMOP data catalogue within the REALM project has been a significant effort, involving technical development, expert collaboration, and alignment across multiple work packages. The journey to design a FAIR data catalogue took some iterations to make it meet our AI evaluation objectives and fit with the REALM architecture components. It was indeed essential to ensure that the catalogue was technically robust, and usable by different end-users through different applications. Each step of the development was carefully discussed, tested, and refined through iterative feedback loops.

Table 5 Outreach and knowledge sharing activities

Event	Date	Role/Contribution	Impact on Catalogue Development
REALM General Meeting 2023	Sept 2023	Consortium-wide alignment – WP & D4.5 Reporting.	Introduced initial need for a standardized data repository.
REALM General Meeting Heraklion	June 2024	Leading parallel session: OMOP mapping of use-cases requirements	Initial introduction of standardization and the OMOP vision – initiate mapping of the use cases requirements and cohort definition to OMOP domains.
OHDSI Europe Symposium 2024	June 2024	2 abstracts submitted and 2 posters presented.	First introduction of REALM and OMOP workflow to the OHDSI community – presentation of the TraqBeat (TB) COPD use-case.
Technical Meeting Brussel	Nov 2024	Highly technical alignment meeting with REALM partners	Demonstration of submitting the TB COPD model in the RIANA dashboard, showcase of the OMOP autocomplete search tool.
REALM General Meeting Warsaw	June 2025	National node working group in-person meeting.	Finalize the agreements context (compensation, access type…) and initiate setup infrastructure of the national nodes.
OHDSI Europe Symposium 2025	July 2025	2 abstracts submitted	Presentation of the OMOP Data catalogue progress, model card and RIANA Dashboard.

The development of the data catalogue has evolved from a conceptual need to a fully operational, integrated component. The integration was significantly accelerated by providing a testing and development server. A minimal configuration of the data catalogue was deployed on a public server, allowing the RIANA dashboard to accessing data through OHDSI WebAPI [3], [4] and creating a demonstration that was presented for the first time during the Brussel technical meeting in November 2024.

3.4 Outreach and knowledge sharing in OHDSI Symposium

Since the initial development of our OMOP-based data workflow, we have consistently reported our progress to the OHDSI community. Starting from the first project deliverable (D4.5, M9), we submitted 5 publications (abstract and posters – see Table 6) to share our methodology to OHDSI Annual Symposium showcasing the evolution of our approach. These contributions not only aim to promote transparency and reuse but also serve to connect with potential data partners and members of various European federated networks.

Table 6 REALM data catalogue and OMOP mapping contributions presented within the OHDSI community

Title	Authors	Year	Description
Harmonizing Public Research Access Datasets for AI Advancements in Asthma/ COPD Diagnosis[5] (Annex 2)	Frederic Jung, Vangelis Sakkalis, Chang Sun, Mahmoud Ibrahim, Gökhan Ertaylan	2024	Describes harmonizing COPD datasets (KaggleCOPD, MIMIC-III) for the evaluation of TraqBeat COPD.
Advancing Certification and Evaluation of Medical Device Software in the EU using OMOP[6] (Annex 2)	Frederic Jung, Chang Sun, Mahmoud Ibrahim, Gökhan Ertaylan	2024	High-level introduction of the REALM project and the OMOP-based evaluation and reporting workflow.
Enhancing Transparency in Healthcare: Blockchain-Supported Model Cards for AI Evaluation Reporting with OMOP and Heracles (Annex 2)	Frederic Jung, Ankur Lohachab, Guilherme Madureira Sanches Ribeiro, Stef Rommes, Visara Urovi, Chang Sun, Gökhan Ertaylan	2025	Explain how REALM is enriching the concept of model card with OMOP for medical AI devices.
RIANA Dashboard: A Blockchain-Enabled Approach for Representing AI Model Intent in OMOP with Atlas-Like Functionality (Annex 2)	Frederic Jung, Guilherme Madureira Sanches Ribeiro, Ankur Lohachab, Stef Rommes, Visara Urovi, Chang Sun, Gökhan Ertaylan, Alfred Attipoe	2025	Introduce the RIANA dashboard and how it was implemented as an alternative to Atlas for capturing patient target groups and map AI features to standard vocabulary.
RAG-Enhanced LLM Pipeline for Semantic Mapping of OMOP-CDM Concepts: Benchmarking AI as Medical Device	Sariga Kakkamani, Frederic Jung, Joeri Verbiest, Liesbet Peeters	2025	Present a use case that goes beyond REALM and the challenges in mapping AI features even when structured.

4 Data Sources

4.1 Adding a new connection

As the REALM platform expands, it is essential to enable access to additional standardized datasets across its federated network. To make a new OMOP CDM data source available within REALM data pool that can be accessed from the RIANA dashboard, a connection needs to be established through WebAPI. Each data source is in a separate schema, with a shared vocabulary, which needs to be linked to WebAPI. Each connection must be registered in two database tables located in the 'OHDSI' WebAPI schema: *source* and *source_daimon (*Figure 5*)*. The *source* table stores the connection details such as database dialect, JDBC URL, and credentials, while the *source_daimon* table defines how that source

[5] OHDSI Symposium 2024 – Poster 16: Harmonizing Public Research Access Datasets for AI Advancements in Asthma/ COPD Diagnosis

[6] OHDSI Symposium 2024 – Poster 104: Advancing Certification and Evaluation of Medical Device Software in the EU using OMOP

is used, whether for CDM data, vocabulary lookup, or results storage. These two tables are linked via the *source_id* key, and each source can have multiple associated daimons (Table 7) specifying its role in the OHDSI framework. Adding a new source to the *ohdsi* schema makes the data source accessible to OHDSI tooling (Table 4) and available for HITL interaction (as described in chapter 2.7, D3.1).

When a new data source is added, a corresponding entry must be created in the source table with the appropriate database access parameters. Then, at least one daimon entry must be added in the *source_daimon* table to tell WebAPI what the source is used for. This setup allows the OHDSI tooling (Table 4) to target the appropriate schema when executing cohort definitions or performing vocabulary queries through WebAPI.

Figure 5: WebAPI Data Source Configuration - This UML diagram illustrates the structure and relationship between the source and source_daimon tables in the WebAPI configuration database. Each row in the source table represents a unique OMOP CDM data source, including its access credentials and connection parameters. The source_daimon table defines the purpose(s) of that source, linked by the shared source_id key.

Table 7 Daimon type reference

Daimon Type (Code)	Purpose	Typical Schema
0	CDM data source	cdm_database_schema
1	Vocabulary source	vocabulary_database_schema
2	Results schema for cohort analyses	results_database_schema

Connection parameter descriptions (to set in the *source* table):

- **Username**: The database username used to authenticate the connection.
- **Password**: The corresponding password for the database user.
- **source_connection**: The full JDBC connection string (e.g., jdbc:postgresql://host:port/dbname) used by WebAPI to establish a connection.
- **source_dialect**: The SQL dialect of the database (e.g., postgresql, sql server, oracle, redshift). This informs WebAPI how to generate SQL compatible with the target system.

As the connection parameters are set, it allows third-party application such as the RIANA dashboard to access all data sources through WebAPI.

4.2 Synthetic data

In cases where real-world data is insufficient, (e.g., due to small sample sizes, underrepresented subpopulations), REALM supports the use of synthetic data to complement or augment existing datasets. To support this, we have developed a suite of synthetic data generators capable of producing different data modalities, including tabular, time-series, and medical imaging data. More details will be reported in Deliverable D4.6.

The generators are trained on 1) publicly available datasets, and partner datasets, where explicit agreements have been made with data providers or controllers allowing use for this purpose. All generative models are stored and executed locally within the REALM node infrastructure. When synthetic data is needed, the generator produces the required samples on demand. Importantly, once the evaluation task is completed, the synthetic data is deleted. Because synthetic data can be generated quickly from trained models, there is no need to store these datasets persistently within the node. Synthetic data can simply be re-generated as needed.

Once the synthetic data is generated, it undergoes the same data mapping process as RWD. This allows synthetic datasets to be seamlessly integrated into the OMOP-based infrastructure and evaluated using the same pipelines and quality framework.

4.3 Mapping the requirements

4.3.1 Medical terminology

Standardized medical terminology is a critical element of semantic interoperability in healthcare applications. In the context of REALM, for effective data integration, especially when dealing with partner-provided models and datasets, it is essential to map all clinical information to standardized vocabularies in OMOP. The primary purpose of medical terminology mapping in this context is to correctly identify the target patient population and clinical variables relevant to a specific AI model provided by user. However, this is often not clearly defined in the documentation or model descriptions. For example, a model may be described that it was trained by the dataset with feature "Number of tobacco pack" or "severity level of COPD" without precise definitions or standard vocabularies. Consequently, it is challenging to identify the correct concept sets or patient target group within testing data sources.

4.3.2 Unstructured Requirements

Each demonstrator in REALM provides a data requirement document describing their AI model and information about the training data. These documents are delivered as free-text Word files, with no or little use of standardized terminology and programmable structure. As a result, interpreting and operationalizing these requirement documents when aiming to automatically map them to the OMOP CDM is challenging and required manual curation.

To address this, we propose a hybrid approach using Name Entity Recognition (NER) to extract relevant medical terms from unstructured text of the documents, followed by Retrieval-Augmented Generation (RAG) to map unstructured clinical text to standardized OMOP concepts (Figure 6). In the first step, NER is applied to extract relevant medical terms from free-text documents. These extracted entities are then processed by a RAG pipeline, which enhances concept mapping through contextual reasoning.

The RAG approach integrates an information retrieval model such as BM25 to identify potential relevant OMOP concept identifiers that are potentially relevant for each extracted term. Subsequently, a Large Language Model (LLM) evaluates and ranks these candidates based on semantic similarity and contextual fit. The system can return the most appropriate set of OMOP concepts in a structured format.

However, since the source document is unstructured text, the NER model may not reliably extract all relevant medical terms. Especially if the medical terms were presented in a non-standard abbreviation or in a lengthy and complex description. Therefore, we proposed to present the requirement of the model in a structured presentation, facilitating effective extraction of medical terminologies. The mapping process will still require human validation to ensure the correct OMOP concepts are applied. This responsibility and effort is initially performed by the model manufacturers, for whom we provide access to OHDSI vocabulary tools such as Athena and Atlas. Subsequently, the HITL process can assist in reviewing and refining these mappings.

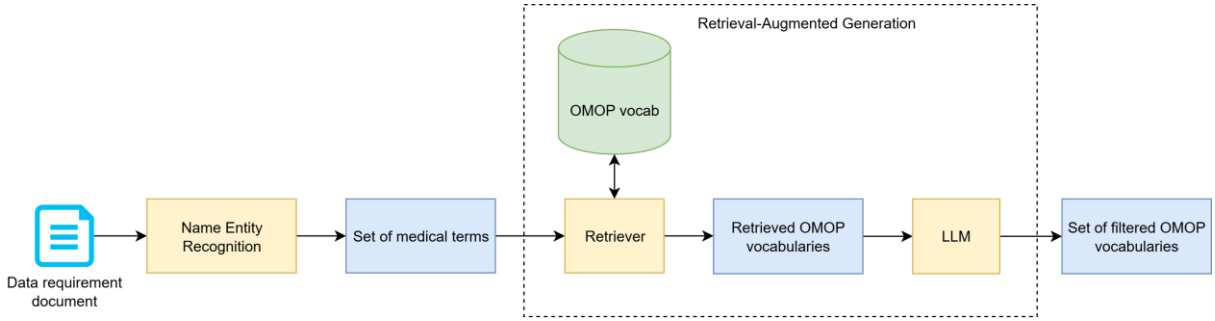


Figure 6: Medical terminology mapping to OMOP process from unstructured data requirement document using Retrieval-augmented generation framework.

4.3.3 Structured Requirements to facilitate mapping

To address the above challenge, we introduced a structured feature definition to represent model input requirements explicitly. It clearly specifies each expected feature, its type, accepted values, and corresponding OMOP concept identifies. An example (below) of how these features can be precisely defined using the examples we mentioned before - "PackHistory" which is a measurement and the number of cigarettes smocked per year for a given patientand "COPDSEVERITY" which is the severity level of COPD. For example, for *PackHistory*, it is defined to be a *Type: numeric, in the Domain: OBSERVATION*, and have a *Description: "Number of cigarettes smoked in a year"*, with a *OMOP Target Concept: 3003421*. This feature defines a quantitative input representing the patient's smoking history, which is mapped directly to an OMOP concept using a single concept ID. By defining this numerically and linking it to the appropriate OMOP concept, we minimize the ambiguity in terminology and interpretation and allow for semi-automated mapping for data-model compatibility

```
"features": [
        {
            "name": "PackHistory",
            "source_values": [],
            "target_values": [
                3003421
            ],
            "description": "Number of cigarettes smocked in a year",
            "type": "numeric",
            "domain": "OBSERVATION",
            "required": true
        },
        {
            "name": "COPDSEVERITY",
            "source_values": [
                "SEVERE",
                "MODERATE",
```

```
            "VERY SEVERE",
            "MILD"
        ],
        "description": "Severity level of Chronic Obstructive Lung Disease",
        "target_values": [
            4209097,
            4193588,
            44791725,
            4196712
        ],

        "type": "categorical",
        "domain": "CONDITION ",
        "required": true
    },
```


4.4 ETL Implementation

To support the transformation of diverse health datasets into the OMOP Common Data Model (CDM), the REALM project has developed a modular and reusable Extract-Transform-Load (ETL) pipeline. This implementation facilitates data harmonization and standardization to the OMOP CDM when needed for the transformation of non OMOP data sources.

4.4.1 ETL scaffolding

The ETL implementation is maintained within a private repository and is built using a Python-based orchestration layer. This orchestration uses SQLAlchemy for database connection and execution of the transformation. Each transformation step is encapsulated in a dedicated SQL script, dedicated Python SQLAlchemy transformation file for each OMOP CDM table that need to be populated. This ensures modularity and easy implementation of the transformations. As the OMOP CDM is a Person centric model, the ETL begins by populating the core Person and Observation Period tables (required later, for cohort definitions generation by the OHDSI WebAPI). Afterward, additional OMOP CDM tables are populated following the order defined by a structural mapping document

4.4.2 Structural Mapping

To ensure traceability and consistency in the structural transformation, we created a structural mapping document template. This template covers all tables from the OMOP CDM v5.4, as well as extensions used in G-CDM (Genomics CDM) and R-CDM (Radiology CDM). This document describes how each field in the source data map to the OMOP CDM as well as the mapping logic. This mapping document serves as both the documentation and the blueprint used by ETL developers to implement the corresponding SQL transformations.

To assist the structural mapping process, the OHDSI community provides dedicated tooling: WhiteRabbit and Rabbit-In-a-Hat. WhiteRabbit performs a scan of the source database and generates a report called a ScanReport that summarize table structures, field contents, and data value distributions in a comprehensive Excel file. The report can then be loaded into Rabbit-In-a-Hat, a graphical tool that enables visual ETL design. It facilitates intuitive table-to-table and field-to-field mapping between the source fields and the OMOP CDM. Both tools can be used to assist analysts and data engineer to fill in the mapping document we created.

4.4.3 Semantic Mapping

Semantic mapping ensures that medical terms, codes, and values from the source data are correctly aligned with the standardized vocabularies of the OMOP CDM v5.4 (vocabulary v20240830 as described in D4.3: Data Management Plan v2). In the OMOP model, every clinical concept is identified

by a unique concept ID from the OMOP standardized vocabulary, which includes terminologies such as SNOMED CT, RxNorm and LOINC. The source codes must be mapped to the OMOP vocabulary and this process is facilitated if the source vocabulary derived from known ontologies which allow us to generate a simple query that find the corresponding standard concept in the hierarchy of the OMOP vocabulary. This process allows us to generate lookup tables that will be integrated into the ETL logic to populate fields in the OMOP tables (such as condition_concept_id, drug_concept_id) and ensuring the semantic consistency across all records within the OMOP database and within the REALM data catalogue. This mapping is critical not only for standardization but also for enabling reliable cross-dataset querying and AI model evaluation within the REALM platform.

In the context of OMOP vocabulary mapping, a source code refers to any clinical code or term used within a data source's native vocabulary system to describe medical events. The source vocabulary reflects the specific encoding used by that data source, whether it relies on an established medical terminology such as SNOMED CT, ICD, or LOINC, or uses entirely custom, non-standard codes or free-text labels unique to the organization. For semantic mapping, the OHDSI tool Usagi is used to assist in mapping source vocabularies to standard OMOP concepts. Usagi loads source codes and, using a term similarity algorithm, suggests mappings to standardized concept IDs from the OMOP vocabulary. However, through practical experience, we observed that Usagi alone is not sufficient to produce high-quality mapping. The task of semantic alignment remains inherently complex and continues to demand significant human supervision. This is primarily due to the nature of the input data, which often includes free-text descriptions, non-standardized codes, or ambiguous clinical expressions that lack consistent structure and terminology.

As a result, Usagi's automated suggestions may include incomplete, imprecise or low-confidence matches, especially for vague or compound terms (e.g., "chronic respiratory issues" or "recent hospitalization"). The tool still relies heavily on manual review by clinical or data experts to verify and refine the suggested mappings (see manual example in Table 8). The effort required for this manual validation increases significantly with the size and variability of the source data. Therefore, the NER and RAG approach described in the chapter above propose an enhanced approach to improve mapping accuracy and will further explore its potential through future work.

Table 8 Subset of semantic mapping for TraqBeat COPD model features

Source Field	Description	Domain	Concept_id
PackHistory	Smoking Pack Years	Observation	3003421
MWT	6MWT Distance	Measurement	606289
FEV1	Forced Expiratory Volume 1	Measurement	4241837
FEV1PRED	Predicted FEV1	Measurement	4208972
FVC	Forced Vital Capacity	Measurement	4176265
FVCPRED	Predicted FVC	Measurement	40482503
CAT	COPD Assessment Test	Measurement	40488830
HAD	Hospital Anxiety and Depression Scale	Measurement	4165145
SGRQ	St. George's Respiratory Questionnaire	Measurement	42872755

4.4.4 The Synthea Example

Synthea is an open-source synthetic patient data generator designed to simulate patient health records based on diseases statistics, clinical guidelines and standard terminologies. It is widely used in academic and research as a privacy-free test dataset for validating healthcare applications and OMOP CDM transformations. Synthea datasets can be downloaded in public official website (https://synthetichealth.github.io/synthea/) for pre-generated versions or users can run Synthea

locally to generate synthetic populations with customized demographics settings. The generated datasets contain a set of CSV files representing tables such as "patient.csv", "encounters.csv", and "conditions.csv", etc. To transform Synthea data into the OMOP CDM, a multi-step ETL process is followed using some OHDSI tools.

The first step involves scanning raw CSV files using WhiteRabbit which helps understand the structure of the source Synthea data. It outputs a detailed scan report containing metadata such as column names, data types, and frequency distributions, etc. For example, running WhiteRabit on "conditions.csv" produces a report showing that the "code" column mostly contains a *SNOMED CT code – 73595000 (Stress).* This scan creates a metadata summary that guiding next mapping step.

Following that, the scan report is loaded into Rabbit-In-a-hat tool. With this tool, each Synthea table can be dragged onto corresponding OMOP tables. For instance, "patient.csv" file is dragged into "PERSON" in OMOP table and "gender" filed in csv file is dragged into "gender_concept_id" field in PERSON table (see Figure 7). Each mapping can include notes about data transformation logics, e.g., how to convert source fields into the target OMOP fields, and these notes can be used to help developers writing the actual ETL codes. These steps are categorized as structural mapping as mentioned in previous Section.

Before executing ETL, OMOP vocabulary tables are downloaded from Athena and will be imported into target OMOP database using a loading script. These vocabularies are needed for mapping standard concept IDs. Once structural mapping and vocabulary are in place, the transformation logic is implemented in Python using Pandas and NumPy libraries. The script can also be written in SQL, which is provided by OHDSI as a reference.

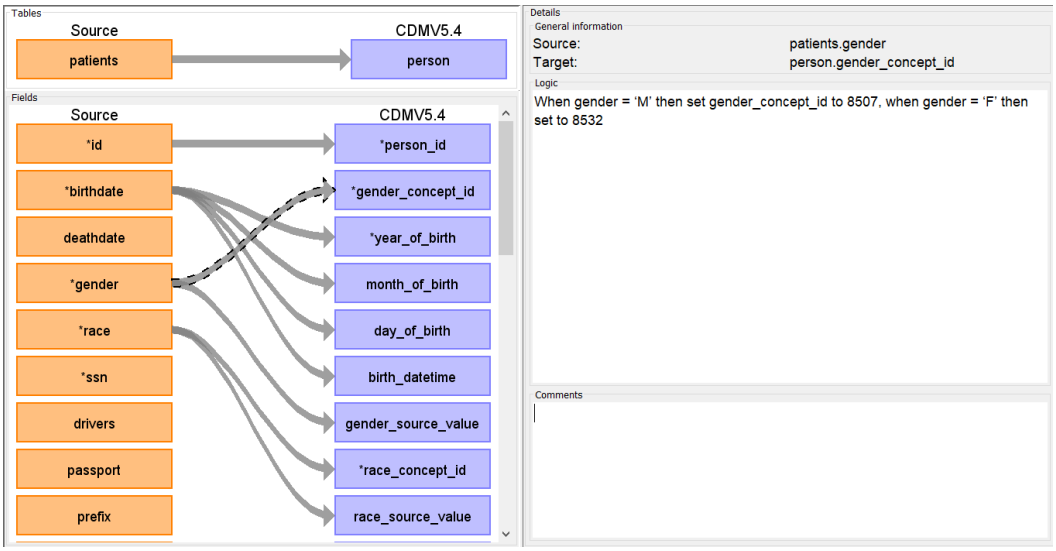


Figure 7: Example of structure mapping for table PERSON using Rabit-In-A-Hat tool.

We applied our methodology on another public patient health dataset, namely the Kaggle COPD dataset[7]. The dataset is a public longitudinal dataset focused on chronic obstructive pulmonary disease, available on Kaggle platform. It includes 101 patients and 24 variables. There's information on their characteristics, disease severity, and co-morbidities, walking ability, quality of life, and anxiety and depression. This dataset was selected not only for its structure and content but also for its relevance to respiratory conditions, making it an ideal preparatory resource for our REALM TraqBeat

[7] COPD dataset on Kaggle Platform: https://www.kaggle.com/datasets/prakharrathi25/copd-student-dataset

demonstrator. We used our ETL scaffolding, to execute the SQL transformations. The Kaggle dataset contains values that mostly map to the domains: *condition*, *observation* and *measurement*.

We are keeping of the structural mapping following our document template, which outlines how each source field maps to the corresponding OMOP CDM field. A first draft of the mapping was done using rabbit in a hat (Figure 8). As the Kaggle COPD dataset is straightforward, the vocabulary mapping was done manually using Athena.

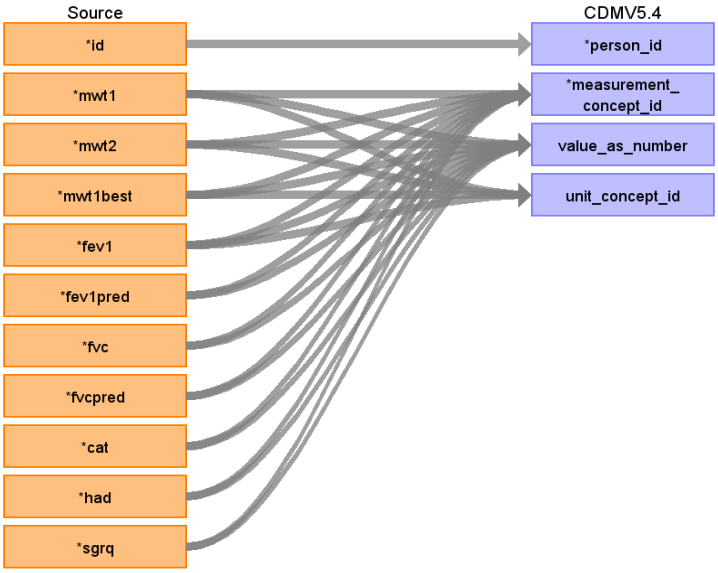


Figure 8: Subset of the structural mapping for table Measurement (OMOP CDM 5.4) generated with Rabbit in a Hat

4.4.5 Mapping of non-tabular file types

While OMOP CDM is primarily designed for structured, tabular healthcare data, in REALM we extended this approach to accommodate non-tabular public data sources such as imaging files (e.g., DICOM), genomic files (e.g., VCF, BAM), and clinical documents (e.g., PDFs, HL7). These sources were mapped using the same principles as traditional tabular data: by adapting our ETL pipelines to populate the appropriate OMOP extensions. For radiology data specifically, we leveraged the OMOP CDM Radiology Extension to store image-derived information. In parallel, we developed a dedicated Python parser that scans directories of DICOM files, flattens the metadata, and prepares it for ETL processing. This ensures that both the imaging metadata and related dataset descriptions are systematically integrated into the OMOP database. This approach was tested for mapping LIDC-IDRI[8] dataset to the OMOP CDM for the evaluation of DuneAI. The data descriptions of each demonstrator were detailed in REALM Deliverable D4.3 (submitted).

Recognizing the need to systematically link supplementary files to patient records within OMOP CDM, we designed and implemented the patient_supplement table (Figure 9 & Annex 3). This REALM-specific OMOP extension provides a structured and standardized way to associate external files with the person table, respecting OMOP's person-centric model and relational integrity constraints. This extension is particularly valuable for demonstrators such as DuneAI and PGx2P, where linking supplementary datasets (e.g., DICOM images, genomic VCF files) to patient records is essential. It enables transparent traceability and simplifies the integration of diverse file types into the REALM evaluation workflow and notably the query builder.

[8] LIDC-IDRI: NIH National Cancer Institute - CIP Cancer Imaging Program
- DOI: 10.7937/K9/TCIA.2015.LO9QL9SX https://www.cancerimagingarchive.net/collection/lidc-idri/v

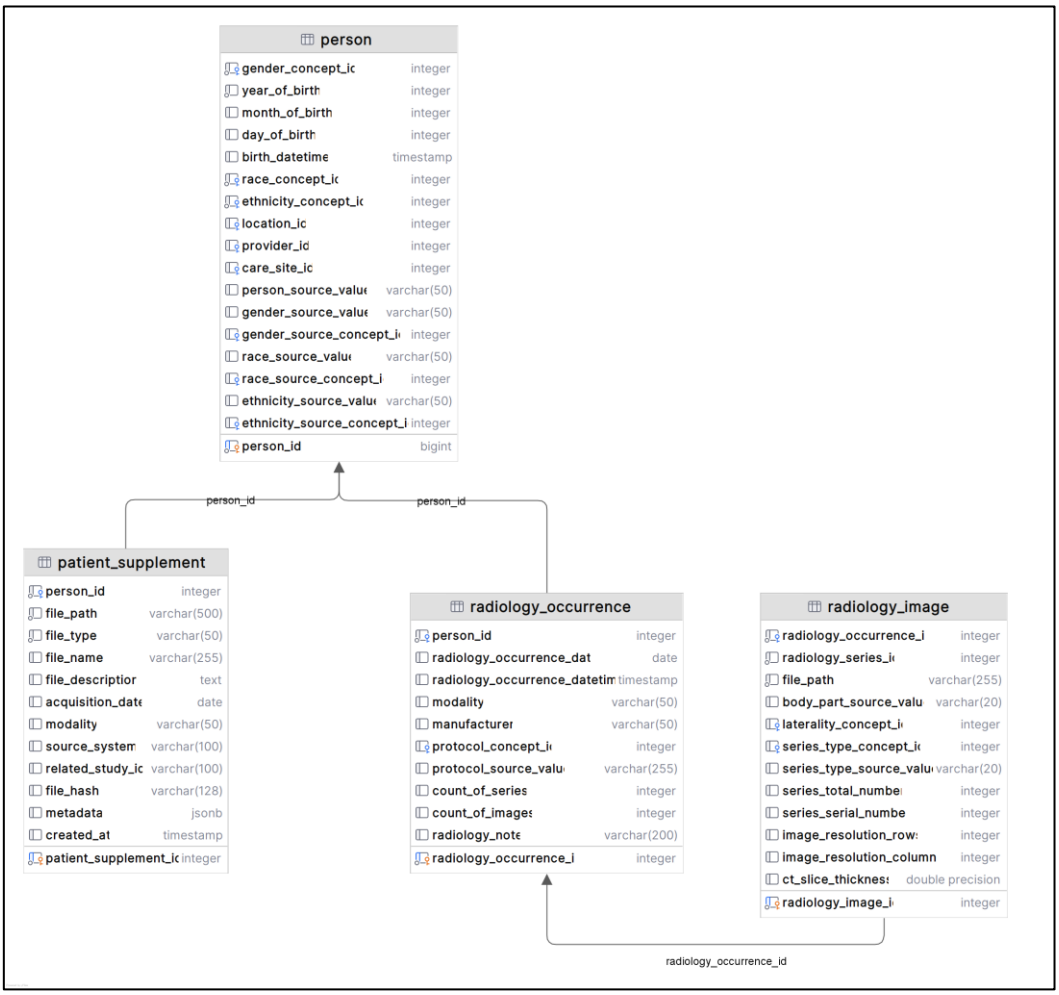


Figure 9: Patient Supplement Table for linking non tabular data files to the OMOP person-centric system.

5 Conclusion

Through the implementation of T4.1, REALM has successfully designed and deployed its federated cloud-based data repository catalogue and infrastructure. REALM's flexible architecture (detailed in D3.1), supports the federated access to real-world health data in a privacy preserving fashion, ensuring alignment with data governance frameworks and national legislation. Documentation and software support to establish a national REALM node has been developed, providing the technical means to interested parties to join the REALM network.

Both direct and indirect data access models are designed. In the direct access model, national nodes host the data locally and allow the evaluation of AI models through local deployment of pipelines. In the indirect access model, evaluation workflows are deployed within the infrastructure of the data provider, thus ensuring that no personal data leaves the owners' institution and that control over sensitive datasets is maintained by the owners. In REALM, only the direct access mode was tested, but the indirect access mode was provided as an alternative configuration. Through the hybrid design, REALM aspires to engage a wider range of data stakeholders, who otherwise might not have been comfortable sharing their data. As part of the minimum viable product, only the direct access model has been released and is currently supported. However, the flexibility of REALM's hybrid design positions the platform to accommodate a broader range of data contributors over time.

The infrastructure builds on the OHDSI tooling ecosystem, which enhances system maintainability and extensibility, but also allows for a tighter integration with the OMOP standard data model. The infrastructure is now capable of supporting key operations such as data ingestion, ETL processing, cohort generation, AI model evaluation via the RIANA dashboard, and generation of standardized model cards.

At the heart of the system is the REALM standardized data catalogue, based on the OMOP data model, and serving as a central reference point for all datasets integrated into the platform. OHDSI tools such as WhiteRabbit, Rabbit-in-a-Hat, and Usagi were used to support data mapping and quality control, although semantic mapping was found to be particularly challenging due to the unstructured nature of model documentation and inconsistent terminology use. To address these issues, a hybrid method combining Named Entity Recognition and Retrieval-Augmented Generation was piloted, with promising results.

The technical developments were demonstrated using both synthetic (Synthea) and real-world (Kaggle COPD) datasets, validating the ETL pipelines and the infrastructure flexibility. Through the process, the need for clear documentation was highlighted, reinforcing the importance of a structured approach to semantic and structural data mapping.

Finally, discussions with data owners revealed that sustainable engagement depends less on financial incentives and more on shared value such as co-authorship opportunities, technical support, access to benchmarking results, and access to the REALM Academy. This insight informed the design of a tailored cooperation model that emphasizes capacity building and long-term strategic collaboration.

In conclusion, Task 4.1 has laid a solid technical and conceptual foundation for the federated REALM data catalogue. The infrastructure is well-aligned with the project's goals of enabling explainable, reproducible, and privacy-preserving AI evaluation in healthcare. The work completed affirms the technical feasibility and strategic relevance of the REALM approach and sets the stage for further validation and scale-up in subsequent phases of the project.

6 Future work

While Task 4.1 successfully delivered the foundational components of the REALM federated data repository, several areas have been identified for further development to ensure long-term impact, sustainability, and integration within the broader EU health data landscape. First, hosting real-world data in REALM National Nodes remains a key barrier, particularly from private healthcare providers and hospital systems. Future efforts can focus on scaling adoption of the indirect access model, which allows data to remain on-site while enabling secure AI model evaluation. This approach reflects similar principles to those adopted by DARWIN EU and should be supported by robust legal frameworks, technical toolkits, and training resources to facilitate participation by a broader range of data partners.

In addition, the European Health Data Space (EHDS) and the establishment of Health Data Access Bodies (HDABs) in each Member State may present a promising opportunity for the use and integration of REALM's national nodes. Given the alignment with privacy-preserving federated architectures, and interoperability standards (e.g. OMOP CDM), REALM nodes are well-positioned to act as complementary technical infrastructures for EHDS. In particular, REALM nodes could serve as specialised AI evaluation environments under the governance of national HDABs to support transparent and reproducible secondary data use for AI evaluation.

Furthermore, to ensure sustainability beyond the project lifecycle, the REALM platform will evolve toward a model that promotes long-term collaboration and shared value for all stakeholders. Future developments can focus on reinforcing REALM as a community-driven infrastructure, offering practical

benefits to participating organisations. These may include access to benchmarking results, structured tooling for AI evaluation, and support and training through the REALM Academy.

7 References

[1] C. Park, S. C. You, H. Jeon, C. W. Jeong, J. W. Choi, and R. W. Park, "Development and Validation of the Radiology Common Data Model (R-CDM) for the International Standardization of Medical Imaging Data," *Yonsei Med J*, vol. 63, pp. S74–S83, Jan. 2022, doi: 10.3349/ymj.2022.63.S74.

[2] R. Belenkaya *et al.*, "Extending the OMOP Common Data Model and Standardized Vocabularies to Support Observational Cancer Research," *American Society of Clinical Oncology*, pp. 12–20, 2021, doi: 10.1200/CCI.20.

[3] Observational Health Data Sciences and Informatics (OHDSI), "OHDSI WebAPI GitHub." [Online]. Available: https://github.com/OHDSI/WebAPI

[4] Observational Health Data Sciences and Informatics (OHDSI), "OHDSI Atlas GitHub." [Online]. Available: https://github.com/OHDSI/Atlas

8 Annex 1: REALM Node Services installation guide

For the following guide, we assume that a server has been configured with the suggested Ubuntu LTS operating system.

8.1 Installing the Services

8.1.1 Docker

Docker is a platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient environments that package an application together with all its dependencies, libraries, and configuration files, ensuring that it runs consistently across different computing environments.

Key Features of Docker

1. **Containerization**: Docker encapsulates applications in isolated containers that include everything needed to run the application, ensuring consistency across development, testing, and production environments.
2. **Portability**: Docker containers can run on any system that has Docker installed, regardless of the underlying infrastructure, making "build once, run anywhere" a reality.
3. **Efficiency**: Unlike traditional virtual machines, Docker containers share the host system's kernel and don't require a full operating system for each application, making them more lightweight and resource-efficient.
4. **Scalability**: Docker makes it easy to scale applications horizontally by spinning up multiple identical containers as needed.
5. **Version Control**: Docker allows versioning of container images, enabling easy rollback to previous versions if issues arise.
6. **Docker Hub**: A cloud-based repository where users can find, share, and store container images, fostering a collaborative ecosystem.
7. **Microservices Architecture**: Docker facilitates breaking down applications into smaller, independent services that can be developed, deployed, and scaled separately.

Components of Docker

- **Docker Engine**: The core of Docker that creates and runs containers
- **Docker Images**: Read-only templates used to create containers
- **Docker Containers**: Running instances of Docker images
- **Docker Compose**: A tool for defining and running multi-container Docker applications

Docker is needed on the REALM node as most of the components that needs to be installed are docker images hosted in a docker image repository.

Suggested method for installation is the convenience script provided by docker.

1. Review the code of the script: https://github.com/docker/docker-install/blob/master/install.sh
2. Download the script: ***$ curl -fsSL https://get.docker.com -o install-docker.sh***
3. Verify the script's content: ***$ cat install-docker.sh***
4. Run the script with ***--dry-run*** to verify the steps it executes: ***$ sh install-docker.sh --dry-run***
5. With root or sudo run the script and follow the steps: ***$ sudo sh install-docker.sh***
6. Verify that the installation is successful by running the hello-world image

$ service docker start

$ docker run hello-world

8.2 Installation of services that use Docker

Now that Docker related software is installed on the server and the docker engine is running, it is time to install the services that rely on Docker. Docker compose is used for the coordination of all installed services. The services that are needed are:

8.2.1 Traefik

A modern HTTP reverse proxy and load balancer designed specifically for microservices and containerized environments. It integrates seamlessly with Docker, Kubernetes, and other infrastructure components, enabling dynamic service discovery and automatic configuration.

Key Features of Traefik

- **Automatic Service Discovery**: Traefik automatically detects services and their changes, reconfiguring itself in real time without requiring restarts.
- **Dynamic Configuration**: It can read its configuration from multiple providers (Docker, Kubernetes, file, etc.) simultaneously.
- **Let's Encrypt Integration**: Traefik provides built-in HTTPS with automatic certificate generation and renewal through Let's Encrypt, as seen in your configuration with certresolver=realmresolver.
- **API and Dashboard**: Includes a comprehensive dashboard for monitoring and management of routes, services, and middleware.
- **Middleware Support**: Offers various middleware for authentication, rate limiting, circuit breaking, and more.
- **Multiple Entrypoints**: Supports different entry points for traffic (HTTP, HTTPS, TCP, UDP), with automatic HTTP to HTTPS redirection as configured in your setup.
- **High Availability**: Designed to be resilient and scalable for production environments.

Traefik is functioning as the gateway to REALM node's application ecosystem, handling all incoming traffic and routing it to the appropriate services Realm node.

No special configuration is needed, all the configuration is included in the docker-compose.yml file that can be found later in this guide.

8.2.2 Zookeeper

A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services for distributed applications. It's a critical component in many distributed systems, particularly those involving Apache Kafka.

Key Features of ZooKeeper

1. **Coordination Service**: ZooKeeper provides primitives such as distributed locks, queues, and barriers that enable reliable coordination between distributed processes.
2. **Configuration Management**: Maintains configuration information and notifies clients of changes, enabling dynamic configuration across distributed systems.
3. **Hierarchical Namespace**: Organizes data in a tree-like structure similar to a file system, with each node (called a "znode") able to store data and have children.
4. **High Availability**: Typically deployed as a cluster (ensemble) to ensure reliability through redundancy.
5. **Consistency**: Guarantees the order of updates and provides a consistent view of the system, even during failures.

6. **Simplicity**: Exposes a simple API that resembles a file system with notifications.

ZooKeeper plays a crucial role in a REALM node by managing the state of Kafka cluster, handling broker coordination, topic configuration, and leader election for topic partitions.

No special configuration is needed, all the configuration is included in the docker-compose.yml file that can be found later in this guide.

8.2.3 Apache Kafka

Apache Kafka is a distributed streaming platform designed for building real-time data pipelines and streaming applications. It provides a unified, high-throughput, low-latency platform for handling real-time data feeds.

Key Features of Kafka

1. **Distributed System**: Scales horizontally across multiple servers for high throughput and fault tolerance.
2. **Pub/Sub Messaging**: Implements a publish-subscribe model where producers send messages to topics and consumers subscribe to topics.
3. **Persistent Storage**: Stores streams of records in a fault-tolerant, durable way with configurable retention periods.
4. **Stream Processing**: Allows for transforming streams of data as they flow through the system.
5. **High Throughput**: Capable of handling millions of messages per second, even with modest hardware.
6. **Fault Tolerance**: Replicates data across multiple nodes to prevent data loss in case of server failures.
7. **Low Latency**: Designed for real-time applications with millisecond latency requirements.

Kafka serves as the central messaging backbone in REALM node, enabling reliable, high-performance data streaming between different components of your REALM application.

8.2.3.1 Kafka certificates generation

Certificates are required for Kafka. These certificates are used for the SSL encryption but also authorization of the clients. So, certificates for both server and clients are needed. The steps to generate them is described below:

8.2.3.2 Generate the .jks certificates for the server

The easiest way to generate and sign you keystore / trustore .jks files is to use a shell script provided by confluent kafka on this https://github.com/confluentinc/confluent-platform-security-tools/blob/master/kafka-generate-ssl.sh) repository.

1. You need to save this script under an empty repo as kafka-generate-ssl.sh
2. You need to have java installed on your wsl
3. Run bash kafka-generate-ssl.sh
4. Follow the step-by-step guide

You will see that a keystore and truststore directory will be created with the keystore.jks and trustore.jks files inside. These are the certificates that we will use to deploy Kafka ssl on our server. Also, this truststore file will be the one that we will use for every other broker, client we will implement.

So now that we are done with our .jks for our server lets generate the certificates for our client.

8.2.3.3 Generate the .jks certificates for our client

Let's move the keystore directory with the keystore.jks file to another directory and rerun bash kafka-generate-ssl.sh. This time to the above question of the script

First, do you need to generate a trust store and associated private key, or do you already have a trust store file and private key? Do you need to generate a trust store and associated private key? [yn]

Answer [n] since we want every other keystore generated to be based on our already generated truststore.

After you follow the steps you have a new keystore.jks files generated in keystore directory. This new keystore.jks and the already generated truststore.jks files are the ones that the client gets. If you have many clients you need to make this procedure once for every client.

Make sure that you rename your keystore.jks files in order not to mix them if you have many clients and of course make sure that you remember your files' passwords

8.2.3.4 Transforming .jks files to pem files

For Connecting to Kafka ssl with Python you don't need truststore/keystore .jks files, but you rather need

- a certificate.pem
- a CARoot.pem
- a key.pem

We are going to generate these 3 files with use only of our keystore.jks file generated on the previous chapter

Copy your clients keystore.jks file to a new empty folder and navigate through wsl to that empty folder

1. First, we need to see the alias of our keystore.jks file with this commant ***$ keytool -v -list -keystore <keystore_name>***
2. Run the above command ***$ keytool -exportcert -alias <alias_name> -keystore <keystore_name> -rfc -file certificate.pem***
3. Run the next command

$ keytool -v -importkeystore -srckeystore <keystore_name> -srcalias <alias_name> -destkeystore cert_and_key.p12 -deststoretype PKCS12

4. Run **$ openssl pkcs12 -in cert_and_key.p12 -nocerts -nodes** This commands only prints the key so you have to copy and paste the text starting with -----BEGIN PRIVATE KEY----- and ending with -----END PRIVATE KEY----- to a new file under the name pey.kem
5. Run ***$ keytool -exportcert -alias CARoot -keystore kafka.keystore.jks -rfc -file CARoot.pem*** to generate the CARoot.pem file.

Except the certificates, no further configuration is needed, all the configuration is included in the docker-compose.yml file that can be found later in this guide.

8.2.4 Kafdrop

A web-based user interface for viewing Kafka topics and browsing consumer groups. It's a lightweight monitoring tool that provides a simple, user-friendly way to examine what's happening in Kafka cluster.

Key Features of Kafdrop

1. **Topic Visualization**: Displays a list of all topics in the Kafka cluster, including their partition count and replication status.
2. **Message Browsing**: Allows you to view messages within topics, with options for filtering and formatting messages in various formats (JSON, Avro, etc.).
3. **Consumer Group Monitoring**: Provides visibility into consumer groups, their lag, and offset information.
4. **Cluster Overview**: Shows broker information and partition distribution across the cluster.
5. **Zero Configuration**: Works out of the box with minimal setup, requiring only a connection to Kafka.
6. **Lightweight**: Runs with minimal resource requirements, making it suitable for both development and production environments.

Kafdrop serves as a visual management interface for Kafka deployment, making it easier to monitor and troubleshoot messaging infrastructure without command-line tools.

No special configuration is needed, all the configuration is included in the docker-compose.yml file that can be found later in this guide.

8.2.5 PostgreSQL

A powerful, open-source object-relational database management system (RDBMS) with a strong reputation for reliability, feature robustness, and performance.

Key Features of PostgreSQL

1. **ACID Compliance**: Ensures Atomicity, Consistency, Isolation, and Durability for all database transactions, maintaining data integrity even during system failures.
2. **Advanced Data Types**: Supports a wide range of built-in and user-defined data types, including JSON, arrays, geometric types, and custom types.
3. **Extensibility**: Allows users to define custom functions, operators, data types, and procedural languages.
4. **Concurrency Control**: Implements Multi-Version Concurrency Control (MVCC) to handle multiple users accessing the database simultaneously without read locks.
5. **SQL Compliance**: Highly compliant with SQL standards while offering powerful extensions.
6. **Robust Security**: Provides column and row-level security, strong authentication mechanisms, and SSL support.
7. **Scalability**: Capable of handling large datasets and high concurrency with tools for replication and partitioning.

PostgreSQL serves as structured data store, managing relational data that requires ACID properties and complex query capabilities.

No special configuration is needed, all the configuration is included in the docker-compose.yml file that can be found later in this guide.

8.2.6 MinIO

A high-performance, S3-compatible object storage system designed for AI, machine learning, and cloud-native workloads. It's built to be simple, scalable, and secure, making it an excellent alternative to traditional storage solutions.

Key Features of MinIO

1. **S3 Compatibility**: Fully implements the Amazon S3 API, allowing applications designed for S3 to work seamlessly with MinIO.
2. **High Performance**: Optimized for performance with capabilities to handle petabytes of data and billions of objects.
3. **Distributed Architecture**: Can be deployed across multiple nodes for horizontal scaling and high availability.
4. **Erasure Coding**: Uses advanced erasure code algorithms to provide data protection against hardware failures and bit rot.
5. **Encryption**: Supports server-side and client-side encryption for data security at rest and in transit.
6. **Identity Management**: Integrates with external identity providers through OpenID Connect and Active Directory.
7. **Bucket Management**: Supports versioning, object locking, retention policies, and lifecycle management.
8. **Web-based Console**: Provides an intuitive management interface for administering buckets, users, and policies.

MinIO serves as unstructured data store, providing a scalable solution for storing and retrieving large amounts of binary data like files, images, and other media that don't fit well in relational databases.

No special configuration is needed, all the configuration is included in the docker-compose.yml file that can be found later in this guide.

8.2.7 Kubernetes (K3D)

K3D (Kubernetes 3D) is a lightweight wrapper to run K3s (a minimalist Kubernetes distribution) in Docker containers. It makes it easy to create and manage multiple Kubernetes clusters on a single machine, primarily for development and testing purposes.

Key Features of K3D

1. **Lightweight**: Creates Kubernetes clusters with minimal resource consumption, suitable for local development machines.
2. **Docker-based**: Runs each Kubernetes node as a Docker container, eliminating the need for virtual machines.
3. **Multi-cluster Support**: Allows running multiple isolated Kubernetes clusters simultaneously on the same host.
4. **Fast Startup**: Creates clusters in seconds rather than minutes, accelerating development workflows.
5. **Registry Integration**: Includes a built-in container registry for easy local image management.
6. **LoadBalancer Support**: Provides an integrated load balancer for exposing services externally.
7. **CLI-focused**: Offers a simple command-line interface for creating, managing, and deleting clusters.
8. **Full Kubernetes API**: Despite being lightweight, provides a complete Kubernetes API that's compatible with standard tools like kubectl.

K3D makes Kubernetes accessible for developers by removing the overhead of managing complex infrastructure, allowing them to focus on application development and deployment.

8.2.7.1 Installation of K3d

Copy the following code to a script called install_k3d.sh

```
#!/bin/sh
set -e

read -p "Please monitor the installation and provide input as required. Press enter to continue..." input

# Get kubectl for kubernetes management
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

# Validate binary
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
read -p "Make sure that the validation check was OK, and press enter to continue, otherwise stop the script CTRL+C and restart it" input

# Install kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# Check succesfull installation
kubectl version --client

# Install K3d
echo "Starting K3d installation"
curl -s https://raw.githubusercontent.com/k3d-io/k3d/main/install.sh | bash
```

Then make the script executable and run it with the following commands:

$ chmod +x install_k3d.sh

$ ./ install_k3d.sh

Then with the following command a cluster is created and ready for use

$ k3d cluster create kfp --api-port 6550 -p "8082:80@loadbalancer" --agents 1 # create a K3D cluster

8.2.8 Kubeflow Pipelines (KFP)

A platform for building and deploying portable, scalable machine learning (ML) workflows based on Docker containers. It's a key component of the larger Kubeflow ecosystem, which aims to make deploying ML workflows on Kubernetes simple, portable, and scalable.

Key Features of Kubeflow Pipelines

1. **Workflow Orchestration**: Allows defining and executing sequences of ML tasks where each step runs in its own container.
2. **Pipeline Composition**: Enables building complex ML workflows using Python SDK or YAML definitions, with components that can be reused across pipelines.
3. **Experiment Tracking**: Automatically tracks pipeline runs, parameters, and metrics for comparison and reproducibility.
4. **Artifact Management**: Manages the inputs and outputs of each pipeline step, including models, datasets, and evaluation results.
5. **Visual Interface**: Provides a web UI for creating, monitoring, and analyzing pipeline runs with visualization of the workflow graph.

6. **Kubernetes Native**: Runs natively on Kubernetes, leveraging its scalability and resource management capabilities.
7. **Integration Capabilities**: Connects with various ML frameworks, data sources, and storage systems through extensible components.
8. **Reproducibility**: Ensures consistent execution environments through containerization, enabling reproducible ML experiments.

Kubeflow Pipelines simplifies the orchestration of complex ML workflows while providing the infrastructure for experiment tracking, reproducibility, and scalability that is essential for modern machine learning development.

8.2.8.1 Installation of KFP

Create a file called install_kfp.sh and copy inside it the following code:

```
#!/bin/sh
set -e

echo "This will deploy Kubeflow Pipelines Standalone instance v2.2.0"

read -p "Please monitor the installation and provide input as required. Press enter to continue..." input
echo ""
read -p " Press enter to continue, or CTRL+C to stop the script..." input
# Do the installation
export PIPELINE_VERSION=2.2.0
kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/cluster-scoped-resources?ref=$PIPELINE_VERSION"
kubectl wait --for condition=established --timeout=60s crd/applications.app.k8s.io
kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/env/dev?ref=$PIPELINE_VERSION"

echo 'Applying ingress for kfp.local.exus.ai access'
kubectl apply -f ./kfp_ingress.yaml

echo 'If no errors occurred installation of Kubeflow pipelines is complete'
echo ""
echo "Kubeflow pipelines will be ready when all pods, running next command, are in ready state"
echo "kubectl -n kubeflow get pods"
```

Then make the script executable and run it with the following commands:

$ chmod +x install_kfp.sh

$ ./ install_kfp.sh

As after running the above commands the deployment is performed by gradually installing all the required pods, the status of the deployment can be checked with the following command:

$ kubectl -n kubeflow get pods

When all pods are in status *Running* the installation is completed.

Last step is to create a file named **kfp_ingress.yaml** and copy the following contents inside it

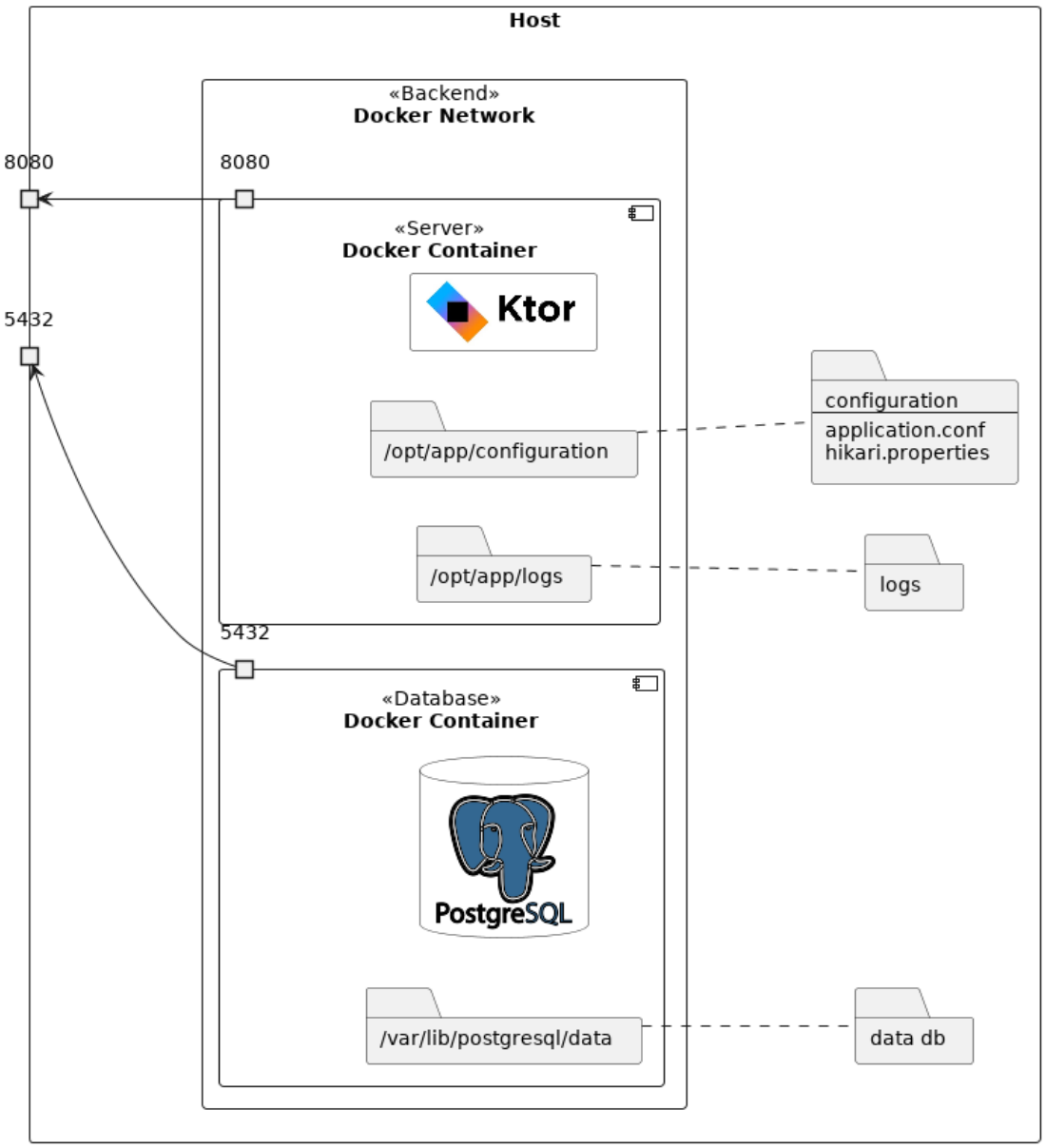
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ml-pipeline-ui-ingress
  namespace: kubeflow
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: your-domain-kfp.ai
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: ml-pipeline-ui
                port:
                  number: 80
```

Apply the above ingress to K3D by running the following command:

$ kubectl apply -f kfp_ingress.yaml

8.3 RIANA Dashboard

REALM users interact through RIANA dashboard with the REALM platform, either to submit a MDSW for evaluation or to follow up with the results of an evaluation. The implementation of the RIANA dashboard is based on a clean multi-tier architecture. At the foundation sits a PostgreSQL database that serves as the persistent data layer, storing all application data with the reliability and ACID compliance that PostgreSQL provides. The front-end application is built with Angular framework, providing an interactive single-page application interface that delivers a responsive user experience through component-based architecture. It incorporates advanced data visualization capabilities using either ChartJS library to render complex datasets into intuitive charts, graphs, and visual representations for dashboard users. The backend consists of a server-side application built with Kotlin, which acts as the business logic and API layer. This Kotlin-based backend processes requests from the frontend, implements business rules, handles authentication and authorization, and manages all database interactions. Kotlin's modern language features and strong type system provide robust server-side development capabilities while maintaining excellent performance and Java Virtual Machine (JVM) compatibility. The complete RIANA Dashboard has been containerised as shown in the following diagram.



8.3.1 Installation of RIANA Dashboard

Prerequisites:

- get an image of the RIANA Dashboard from Comunicare
- CARoot.pem: file received from Exus
- test.pem: file containing the **concatenation** of key.pem and certificate.pem received from Exus

1. Create a folder containing the configuration files of the server (by example docker-configuration)
   - application.conf: configuration file for the riana-server. The one used for testing can be found in the repo: /src/test/resources/application.conf. **When deployed on a server,**
     - **all the secret keys must be modified!**
     - **host: 0.0.0.0**
   - hikari.properties : configuration file for the database

```
dataSourceClassName=org.postgresql.ds.PGSimpleDataSource
```

```
dataSource.databaseName=postgres

dataSource.portNumber=5432
```

consumer.properties : configure file for the kafka consumer

```
group.id=realm-test-group
#concatenation of key.pem and certificate.pem
ssl.keystore.location=/opt/app/configuration/test.pem
bootstrap.servers=realm.exus.ai\:29093
ssl.truststore.type=PEM
ssl.keystore.type=PEM
security.protocol=SSL
enable.auto.commit=false
ssl.truststore.location=/opt/app/configuration/CARoot.pem
auto.offset.reset=earliest
ssl.endpoint.identification.algorithm=
```

producer.properties : configure file for the kafka producer

```
security.protocol=SSL
ssl.keystore.type=PEM
# concatenation of key.pem and certificate.pem
ssl.truststore.location=/opt/app/configuration/CARoot.pem
ssl.keystore.location=/opt/app/configuration/test.pem
bootstrap.servers=realm.exus.ai\:29093
ssl.truststore.type=PEM
ssl.endpoint.identification.algorithm=
```

2. have a folder in which to store the database data
3. have a folder in which to store the server logs (by example logs-docker)
4. have a folder in which to store the test database for unit test
5. Create a file .env containing the variables needed to configure the image

```
POSTGRES_VERSION=16.1

POSTGRES_USER=***

POSTGRES_PASSWORD=***

POSTGRES_DB=postgres

HIKARI_FILE=/opt/app/configuration/hikari.properties

SERVER_IMAGE_NAME=riana-server

SERVER_IMAGE_VERSION=***   (depends   on   the   image   of   the   riana   server)
HOST_SERVER_PORT=8080

HOST_POSTGRES_PORT=5432

HOST_POSTGRES_DATA=*** (see 2)

HOST_POSTGRES_DATA_TEST=*** (see 4)
```

```
HOST_SERVER_LOGS_FOLDER=*** (see 3)

HOST_SERVER_CONFIGURATION=***                    (see                    1)
PRODUCER_FILE=/opt/app/configuration/producer.properties
CONSUMER_FILE=/opt/app/configuration/consumer.properties
```

Replace *** by correct values, depending on the environment

6. Create the containers

```
docker compose up -d
```

7. Show logs

```
docker compose logs -f --since 1m
```

8. List of docker compose in operation

```
docker compose ls
```

9. Getting inside a container

```
docker compose exec riana-server-compose /bin/bash
```

10. Stop

```
docker compose stop
```

8.4 Docker compose file for deployment

After finishing with all the previous steps, it is time to deploy the docker-compose.yml file that deploys and handles the services.

Steps for the installation:

- Create a folder and cd into it.
- Create a ***.env*** file and copy inside the following contents, after adjusting the parameters

```
COMPOSE_PROJECT_NAME=realm

GENERIC_DOMAIN_SUFFIX=your-domain.com

POSTGRES_DB=realm_db
POSTGRES_USER=***
POSTGRES_PASSWORD=***
POSTGRES_PORT_EXTERNAL=5832

MINIO_ROOT_USER=***
MINIO_ROOT_PASSWORD=***
```

Now create the docker-compose.yml in the same folder with the .env and copy inside the following contents, after making again any changes needed to match the domains chosen for the node and also the proper path for the ssl certificates of kafka that has been created previously:

```
---
version: '3'

volumes:
  traefik-acme:
  zookeeper-data:
  zookeeper-logs:
  kafka-data:
  postgres-data:
  minio_data:

networks:
  realm:
    driver: bridge
  kafka:
    driver: bridge

services:
  traefik:
    build: ./traefik
    restart: unless-stopped
    command:
      - "--log.level=INFO"
      - "--api=true"
      - "--api.dashboard=true"
      - "--api.insecure=false"
      - "--providers.docker=true"
      - "--providers.docker.exposedByDefault=false"
      - "--providers.docker.network=${COMPOSE_PROJECT_NAME}_realm"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.web.http.redirections.entryPoint.to=websecure"
      - "--entrypoints.web.http.redirections.entryPoint.scheme=https"
      - "--entrypoints.websecure.address=:443"
      - "--entrypoints.websecure.http.tls=true"
      - "--entrypoints.websecure.http.tls.certresolver=realmresolver"
      - "--certificatesresolvers.realmresolver.acme.email=your@email.com"
      - "--certificatesresolvers.realmresolver.acme.tlsChallenge=true"
      - "--certificatesresolvers.realmresolver.acme.storage=/acme/acme.json"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - traefik-acme:/acme
    networks:
      - realm
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.dashboard.entrypoints=web, websecure"
      - "traefik.http.routers.dashboard.rule=Host(`${GENERIC_DOMAIN_SUFFIX}`)"
      - "traefik.http.routers.dashboard.tls=true"
      - "traefik.http.routers.dashboard.tls.certresolver=realmresolver"
      - "traefik.http.routers.dashboard.service=api@internal"
      - "traefik.http.routers.dashboard.middlewares=admin-auth"
      -                                          "traefik.http.middlewares.admin-
auth.basicauth.usersfile=/etc/traefik/users"
      - "traefik.http.middlewares.admin-auth.basicauth.removeheader=true"
```

```
  zookeeper:
    image: confluentinc/cp-zookeeper:7.4.0
    restart: unless-stopped
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_INIT_LIMIT: 5
      ZOOKEEPER_SYNC_LIMIT: 2
      ZOOKEEPER_SERVERS: localhost:22888:23888
    networks:
      - kafka
      - realm
    volumes:
      - zookeeper-data:/var/lib/zookeeper/data
      - zookeeper-logs:/var/lib/zookeeper/log

  kafka:
    image: confluentinc/cp-kafka:7.4.0
    user: "1001"
    restart: unless-stopped
    ports:
      - "9092:9092"
      - "9093:9093"
      - "29093:29093"
      - "29092:29092"
    environment:
      KAFKA_BROKER_ID: 1
      ZOOKEEPER_SASL_ENABLED: 'false'
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:9092,INTERNAL://kafka:9093,SSL://localhost:29092,EXT://your-
domain.com:29093
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,INTERNAL:SSL,SSL:SSL,EXT:SSL
      KAFKA_SSL_CLIENT_AUTH: required
      KAFKA_SSL_KEYSTORE_FILENAME: kafka.server.keystore.jks
      KAFKA_SSL_KEYSTORE_CREDENTIALS: password.txt
      KAFKA_SSL_KEY_CREDENTIALS: password.txt
      KAFKA_SSL_TRUSTSTORE_FILENAME: kafka.truststore.jks
      KAFKA_SSL_TRUSTSTORE_CREDENTIALS: password.txt
      KAFKA_SSL_ENDPOINT_IDENTIFICATION_ALGORITHM: ""
      KAFKA_SECURITY_PROTOCOL: SSL
      KAFKA_NUM_PARTITIONS: 3
      KAFKA_OFFSETS_TOPIC_NUM_PARTITIONS: 3
      KAFKA_DELETE_TOPIC_ENABLE: 'true'
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: 'true'
      KAFKA_OFFSETS_COMMIT_TIMEOUT_MS: 60000
      KAFKA_GROUP_MAX_SESSION_TIMEOUT_MS: 60000
      KAFKA_LOG4J_ROOT_LOGLEVEL: WARN
    networks:
      - kafka
      - realm
    depends_on:
      - zookeeper
    healthcheck:
      test: nc -z localhost 9092 || exit -1
```

```
      start_period: 15s
      interval: 5s
      timeout: 10s
      retries: 10
    volumes:
      - /etc/kafka:/etc/kafka/secrets
      - kafka-data:/var/lib/kafka/data

  kafka-ui:
    image: obsidiandynamics/kafdrop:4.0.1
    ports:
      - "9000:9000"
    environment:
      KAFKA_BROKERCONNECT: "kafka:9092"
    networks:
      - kafka
      - realm
    depends_on:
      kafka:
        condition: service_healthy
    restart: unless-stopped
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.kafka-ui.entrypoints=web, websecure"
      -                          "traefik.http.routers.kafka-ui.rule=Host(`kafka-
ui.${GENERIC_DOMAIN_SUFFIX}`)"
      - "traefik.http.routers.kafka-ui.tls=true"
      - "traefik.http.routers.kafka-ui.tls.certresolver=realmresolver"
      - "traefik.http.routers.kafka-ui.service=kafka-ui"
      - "traefik.http.routers.kafka-ui.middlewares=admin-auth"
      - "traefik.http.services.kafka-ui.loadbalancer.server.port=9000"

  postgres:
      image: postgres:16.2-bookworm
      environment:
        - POSTGRES_DB
        - POSTGRES_USER
        - POSTGRES_PASSWORD
      ports:
        - "${POSTGRES_PORT_EXTERNAL}:5432"
      networks:
        - kafka
        - realm
      restart: unless-stopped
      volumes:
        - postgres-data:/var/lib/postgresql/data/

  adminer:
    image: adminer:4.8.1-standalone
    depends_on:
      - postgres
    ports:
      - 8080:8080
    networks:
      - kafka
      - realm
    restart: unless-stopped

  minio:
```

```
    image: minio/minio:RELEASE.2025-01-20T14-49-07Z
    ports:
      - "9002:9000"
      - "9001:9001"
    environment:
      MINIO_ROOT_USER: ***
      MINIO_ROOT_PASSWORD: ***
    volumes:
      - minio_data:/data
    command: server /data --console-address ":9001"
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "mc", "ready", "local"]
      interval: 5s
      timeout: 5s
      retries: 5
```

Being in the folder with the docker-compose.yml and .env files, run the following command to start the services:

$ docker compose up -d

All the services should be up and running now.

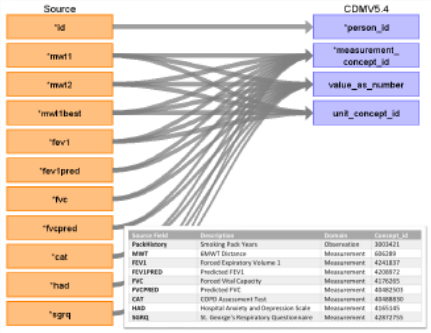
9 Annex 2: OHDSI Posters

Open research datasets harmonized to the OMOP CDM can be used to evaluate COPD AI models.

*Title: **Harmonizing Public Research Access Datasets for AI Advancements in Asthma/ COPD Diagnosis.***

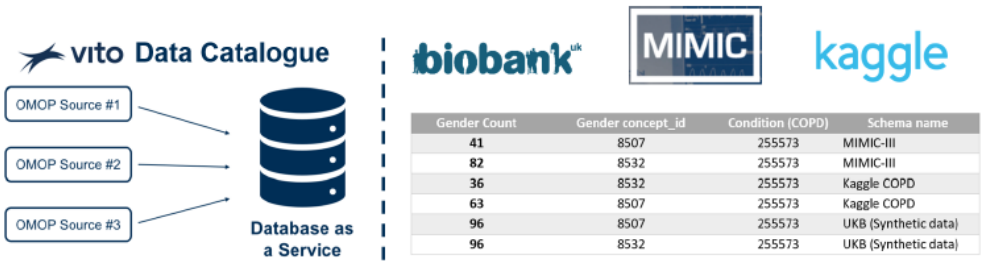
Background: Chronic Obstructive Pulmonary Disease (**COPD**) is a significant global health concern, causing substantial morbidity and mortality worldwide. With millions affected, COPD necessitates advanced diagnostic and therapeutic approaches. Leveraging **Artificial Intelligence (AI)** on standardized datasets offers promise in enhancing COPD management. Our study focuses on **evaluating three research datasets (Kaggle COPD datasets, MIMIC3, UK Biobank)**, harmonized to the Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM), to develop and assess advanced AI solutions for COPD. Traqbeat, a medical technology company, provided two indicative COPD AI models for demonstration, illustrating the benefits and challenges of AI model training, testing, and assessment with OMOP.

Figure1: Structural and Semantic mapping.



The mapping approach undertaken in this study involves both structural and conceptual mapping to harmonize COPD-related datasets to the OMOP CDM (*Figure 1*) ensuring **consistency and interoperability**. The study focused on mapping specific **clinical measurements** and variables relevant to COPD diagnosis and management. Additionally, the implementation of **Extract, Transform, and Load (ETL)** processes was crucial in translating these mappings into practical implementation.

Figure2: High level summary of the COPD population in the VITO database.



Gender Count	Gender concept_id	Condition (COPD)	Schema name
41	8507	255573	MIMIC-III
82	8532	255573	MIMIC-III
36	8532	255573	Kaggle COPD
63	8507	255573	Kaggle COPD
96	8507	255573	UKB (Synthetic data)
96	8532	255573	UKB (Synthetic data)

Conclusion: Harmonizing datasets to OMOP ensures consistency in clinical data representation, facilitating cross-database analyses (*Figure 2*) and promoting AI solutions for COPD diagnosis and management. By utilizing public research datasets, we eliminate complex data access agreements, fostering collaboration and encouraging broader participation in developing solutions for COPD research

Frédéric Jung[1], Vangelis Sakkalis[2], Chang Sun[3], Mahmoud Ibrahim[3], Gökhan Ertaylan[1]

1) VITO, Vlaamse Instelling voor Technologisch Onderzoek, Mol, Belgium
2) Traqbeat technologies, Heraklion, Greece
3) Institute of Data Science, Maastricht University, Maastricht, the Netherlands

REALM – innovative solutions for the creation and **evaluation** of **AI medical device software.**

*Title: **Advancing Certification and Evaluation of Medical Device Software in the EU using OMOP.***

Funded by the European Union

Background: In the landscape of healthcare, the increasing complexity and autonomy of **Medical Device Software (MDS)** present significant challenges in their certification and post-market monitoring, particularly in adapting to **real-world** healthcare settings. To address these challenges, the REALM project, also known as Real-world-data enabled assessment for health regulatory decision-making, aims to provide a robust testing infrastructure for the **evaluation** and certification of MDS in the European healthcare industry. By emphasizing transparency through the use of **OMOP** (Observational Medical Outcomes Partnership) databases and extensions, REALM seeks to offer stakeholders and regulatory bodies detailed and transparent insights into the performance of **AI** models embedded in MDS, crucial for ensuring trust and reliability in medical software solutions.

Figure1: REALM Partners across Europe.

Figure2: REALM interoperability vision & harmonization plan.

REALM is a significant collaborative effort involving 15 partners across Europe (*Figure 1*). Within this consortium, five partners act as **demonstrators**, providing **AI** models for evaluation of the REALM capabilities to generate relevant testing dataset and evaluate how **AI** models performed. Given the **diversity** of data types required by the demonstrators (and future AI models) REALM heavily rely on the **OMOP CDM** and Extensions (**R-CDM** - Park et al., 2022 & **G-CDM** - Shin et al., 2019) to harmonize source data (*Figure 2*) from the **REALM data catalogue** and provide a unique way to query and generate test datasets. Following this approach, REALM also plans to incorporate data from the embedded **synthetic data generator** and **digital human twin** into its evaluation framework. By integrating these additional resources, REALM aims to further enhance the diversity and richness of its testing datasets, providing a more comprehensive **assessment of AI model performance across various healthcare scenarios.**

Conclusion: REALM's integration of diverse data sources using OMOP CDM together with the REALM's rigorous framework provide **regulatory bodies** with a powerful **sandbox environment** for a transparent and precise assessment of medical AIs.

REALM
EU Project n°101095435

Frédéric Jung[1], Chang Sun[2], Mahmoud Ibrahim[2], Gökhan Ertaylan[1]

1) VITO, Vlaamse Instelling voor Technologisch Onderzoek, Mol, Belgium

2) Institute of Data Science, Maastricht University, Maastricht, the Netherlands

Enhancing transparency in AI for healthcare with OMOP Model Card reporting

Title: ***Enhancing Transparency in Healthcare - Blockchain-Supported Model Cards for AI Evaluation Reporting with OMOP and Heracles.***

Funded by the European Union

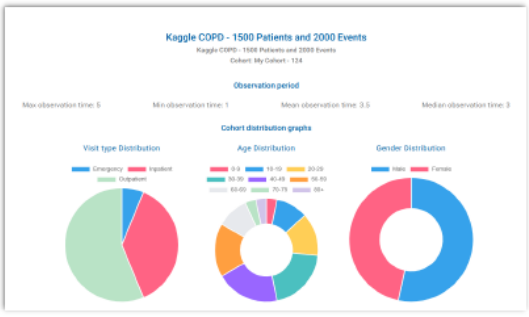
AI models in healthcare demand rigorous documentation to support transparency, trust, and regulatory alignment—particularly under evolving EU regulations like the AI Act and EHDS. We propose an extended model card framework, rooted in OHDSI methodologies and the OMOP CDM, that standardizes the way AI models intent and performance are described. This framework clearly defines target cohorts, input features, evaluation metrics, and data requirements using structured JSON formats linked to OMOP concepts, enabling semantic clarity and consistent model documentation across OMOP-CDM-compliant clinical datasets used for AI model development and evaluation.

Table 1: Structured Model card sections aligned with OMOP CDM

Model Card Section	Representative Examples
Factors Includes demographic or phenotypic groups, environmental conditions, and other stratification factors. These are defined as cohort definitions in JSON format, allowing a structured representation of patient groups.	"InclusionRules": [{ "expression": { "Gender": { "CONCEPT_ID": 8507, "CONCEPT_NAME": "MALE", } }, "ConditionOccurrence": { "CodesetId": 12, "Age": { "Value": 22, "Op": "gt" }
Features Provides the feature mapping between the input dataset and the OMOP CDM. For each feature and its modalities, the mapping ensures alignment with standardized OMOP concept.	{ "name": "COPDSEVERITY", "source_values": ["SEVERE", "MODERATE",], "target_values": [4209097, 4193588,], "type": "categorical", "domain": "condition_occurrence", },
Evaluation Dataset Provides details on the dataset used for quantitative analyses in the model card. This includes Heracles results per OMOP data sources along with patient counts, generation timestamps, summarizing demographics and clinical distributions	"datasets": [{ "name": "MIMICIII", "datasetID": "1234", "summary": { "PersonDistribution": { "conceptName": "MALE", "conceptId": 8507, "countValue": 95 }, "ConditionDistribution": [{ "conceptId": 4193588, "conceptPath": Moderate COPD", "numPersons": 39}, { "conceptId": 4209097, "conceptPath": Severe COPD", "numPersons": 27 },{…}
Metrics Reflects the potential real-world impact of the model by capturing performance measures. These metrics are stored alongside the evaluation dataset and are linked using the same *datasetID*.	"metrics": [{ "datasetID": "1234", "name": "Accuracy", "score": "0.85", }, { "datasetID": "1234", "name": "Recall", "score": "0.92", }],

Table 1 shows how OMOP-aligned JSON fields structure the card, while Figure 1 gives an example of it can be visualized. To ensure traceability and auditability, model cards are published as NFTs on the Polygon blockchain using the ERC-1155 standard. The RIANA dashboard, a web-based interface, guides model developers to create the model card through a 4-step process: (1) cohort definition via WebAPI, (2) cohort generation across OMOP sources, (3) Heracles-driven summary analysis, and (4) model card generation and blockchain publication. Each version is immutable, timestamped, and linked to data source provenance, supporting regulatory and institutional review processes.

Figure 1: Model Card Visualization Component .



Each model card encapsulates performance metrics and cohort characteristics in a format interpretable by OHDSI tools such as Atlas. This framework supports manufacturers in regulatory approval processes (e.g., CE marking) and fosters an ecosystem of reproducible and trustworthy medical AI.

REALM

EU Project n°101095435

Frédéric Jung[1], Ankur Lohachab[2], Guilherme Madureira Sanches Ribeiro[3], Stef Rommes[1], Visara Urovi[2], Chang Sun[2], Gökhan Ertaylan[1]

(1) VITO, Vlaamse Instelling voor Technologisch Onderzoek, Mol, Belgium (2) Institute of Data Science, Maastricht University, Maastricht, the Netherlands (3) COMMUNICARE Solutions, Liege, Belgium

vito

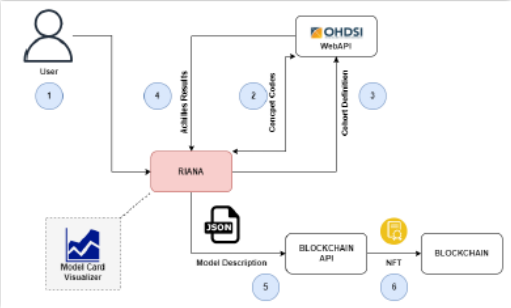
A user-friendly alternative to **OHDSI Atlas** for **documenting** and **reporting** medical **AI models** claims

Title: ***RIANA Dashboard: A Blockchain-Enabled Approach for Representing AI Model Intent in OMOP with Atlas-Like Functionality***

Funded by the European Union

AI models in healthcare face increasing demands for transparency, reproducibility, and regulatory alignment. To meet these needs, we developed the **REALM Intelligent Analytics (RIANA) Dashboard**, an alternative user interface to Atlas specifically designed to capture and formalize the intent behind AI models used in clinical practice. RIANA helps users to define patient cohorts, map input features and report performance metrics using standardized vocabularies and clinical concepts from the OMOP Common Data Model. This structured approach ensures that each AI model is clearly linked to a well-defined target population, streamlining preparation for CE marking and compliance with emerging EU regulatory frameworks.

Figure 1: High-level user workflow in the RIANA dashboard



RIANA simplifies the process of defining and documenting the claims of AI models through an intuitive web-based interface. Model intended purpose and input features can be easily mapped to the standardized OMOP vocabulary (Figure 3).

Figure 1: Based on this structured input, RIANA users can easily search for concept codes ❶ ❷ to generate cohort definitions ❸ translating inclusion and exclusion criteria directly from the model's intended use. Detailed concept distributions and demographics for each scenarios are generated from Achilles ❹.

Figure 2: Simplified RIANA dashboard reporting workflow using WebAPI and Blockchain API

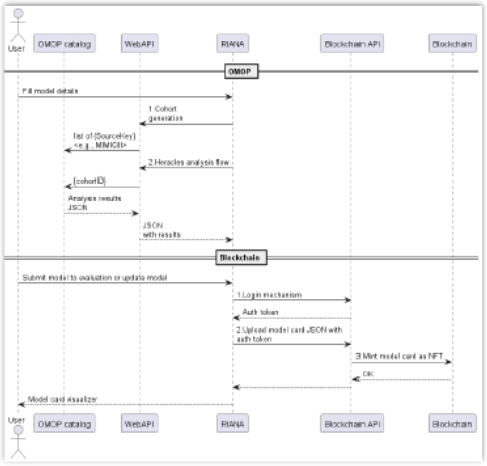
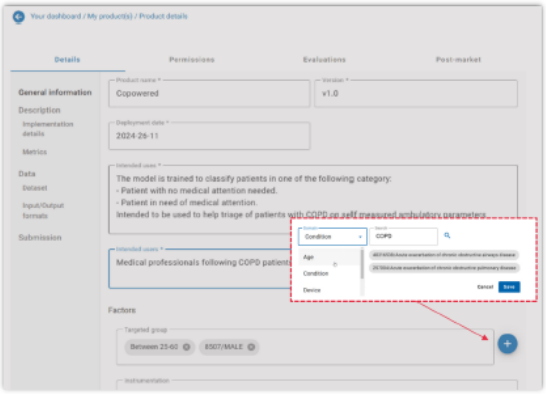


Figure 3: RIANA Dashboard Interface with OMOP Autocomplete for Defining Patient Target Group



Finally, all components, including the cohort definition ❸, Achilles outputs ❹, and performance metrics, are bundled into a structured JSON model card ❺, providing a transparent and reproducible summary that can also be visualized and published ❻ for data partners, regulatory authorities and notify bodies.

REALM
EU Project n°101095435

Frédéric Jung[1], Guilherme Madureira Sanches Ribeiro[2], Ankur Lohachab[3], Stef Rommes[1], Visara Urovi[3], Chang Sun[3], Gökhan Ertaylan[1] , Alfred Attipoe[2]

1) VITO, Vlaamse Instelling voor Technologisch Onderzoek, Mol, Belgium (2) COMMUNICARE Solutions, Liege, Belgium (3) Institute of Data Science, Maastricht University, Maastricht, the Netherlands

vito

10 Annex 3: REALM extension for non-tabular data sources