Artificial intelligence Supporting CAncer Patients across Europe

Project Title	Artificial intelligence Supporting CAncer Patients across Europe
Project Acronym	ASCAPE
Grant Agreement No	875351
Instrument	Research and Innovation action
Call / Topic	H2020-SC1-DTH-2019 / Big data and Artificial Intelligence for monitoring health status and quality of life after the cancer treatment
Start Date of Project	01/01/2020
Duration of Project	36 months

D3.1 – Cancer-care predictive analytics and decision-making services: proof of concept demonstration

Work Package	WP3 – Evidence-based Health & Well-being Monitoring
Lead Author (Org)	Mirjana Ivanović (UNSPMF)
Contributing Author(s) (Org)	Vladimir Kurbalija (UNSPMF), Miloš Savić (UNSPMF), Brankica Bratić (UNSPMF), Nataša Vujnović Sedlar (UNSPMF), Mihailo Ilić (UNSPMF), Serge Autexier (DFKI), Johannes Rust (DFKI), Lucian Itu (SIE), Anamaria Vizitiu (SIE), Radu Toev (SIE), Tzortzia Koutsouri (STS), Konstantina Koloutsou (STS) Antonis Valachis (ORB), Miltiadis Kokkonidis (INTRA), Thanos Kosmidis (CC), Manuel Perez (ATOS), Cristina Sabater Useros (ATOS), Kostas Perakis (UBITECH), Dimitris Miltiadou (UBIITECH)
Due Date	28.02.2021
Actual Date of Submission	27.02.2021
Version	2.0

The work described in this document has been conducted within the project ASCAPE. This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 875351. This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

Dissemination Level

X	PU: Public (*on-line platform)
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

Versioning and contribution history

Version	Date	Author	Notes
0.1	06.11.2020	Mirjana Ivanović (UNSPMF)	Initial proposal for the Table of Contents
0.2	15.11.2020	Mirjana Ivanović (UNSPMF), Serge Autexier (DFKI), Johannes Rust (DFKI), Lucian Itu (SIE), Tzortzia Koutsouri (STS), Konstantina Koloutsou, (STS), Antonis Valachis (ORB), Vladimir Kurbalija (UNSPMF), Miloš Savić (UNSPMF), Brankica Bratić (UNSPMF), Miltiadis Kokkonidis (INTRA), Thanos Kosmidis (CC), Manuel Perez (ATOS)	Second version of the Table of Contents
0.3	16.12.2020	Mirjana Ivanović (UNSPMF), Serge Autexier (DFKI), Johannes Rust (DFKI), Lucian Itu (SIE), Anamaria Vizitiu (SIE), Tzortzia Koutsouri (STS), Konstantina Koloutsou (STS), Antonis Valachis (ORB), Vladimir Kurbalija (UNSPMF), Miloš Savić (UNSPMF), Brankica Bratić (UNSPMF), Mihailo Ilić (UNSPMF), Miltiadis Kokkonidis (INTRA), Thanos Kosmidis (CC), Manuel Perez (ATOS), Cristina Sabater Useros (ATOS), Kostas Perakis (UBITECH)	First textual input and reports on demos
0.4	14.01.2021	Mirjana Ivanović (UNSPMF), Serge Autexier (DFKI), Johannes Rust (DFKI), Lucian Itu (SIE), Anamaria Vizitiu (SIE), Tzortzia Koutsouri (STS), Konstantina Koloutsou (STS), Vladimir Kurbalija (UNSPMF), Miloš Savić (UNSPMF), Brankica Bratić (UNSPMF), Mihailo Ilić (UNSPMF), Miltiadis Kokkonidis (INTRA), Manuel Perez (ATOS),	First complete version of deliverable/draft

Version	Date	Author	Notes
		Cristina Sabater Useros (ATOS), Kostas Perakis (UBITECH)	
1.0	05.02.2021	Mirjana Ivanović (Summary, Introduction, Conclusion, UNSPMF), Serge Autexier (DFKI), Johannes Rust (DFKI), Lucian Itu (SIE), Anamaria Vizitiu (SIE), Radu Toev (STS), Tzortzia Koutsouri (STS), Konstantina Koloutsou (STS), Vladimir Kurbalija (UNSPMF), Miloš Savić (UNSPMF), Brankica Bratić (UNSPMF), Miltiadis Kokkonidis (INTRA), Manuel Perez (ATOS), Cristina Sabater Useros (ATOS), Kostas Perakis (UBITECH), Dimitris Miltiadou (UBIITECH)	Second version
1.1	25.02.2021	All contributing authors	Corrections to address issues identified during the first internal review
2.0	26.02.2021	Mirjana Ivanović (UNSPMF)	Final version

Table of Contents

List of tables

List of figures

List of Acronyms

ADAB	Adaptive Boosting (AdaBoost)
AI	Artificial Intelligence
CLI	Command-Line Interface
CSV	Comma-separated Values
DP	Differential Privacy
DT	Decision Tree
FHIR	Fast Healthcare Interoperability Resources
HE	Homomorphic Encryption
HIS	Hospital Information Systems
HL7	Health Level 7
IIEF	International Index of Erectile Function
INC	Incremental
IPSS	International Prostate Symptom Score
KNN	K-Nearest-Neighbour
LIFT	Deep Learning Important FeaTures
LISAT	Life-Satisfaction Questionnaire
MAE	Mean Absolute Error
MaaS	Model as a Service
ML	Machine Learning
MLP	Multilayer Perceptron
MORE	Matrix Operation for Randomization or Encryption
MSE	Mean Squared Error
MVI	Missing Values Inference
NB	Naïve Bayes
NN	Neural Network
OCI	Open Container Initiative
OIDC	Open Id Connect
ORB	Örebro
PoC	Proof of Concept
QoL	Quality of Life
RF	Random Forest
SDK	Software Development Kit
SGD	Stochastic Gradient Descent
SKLR	Scikit-Learn Based Regression
SHAP	SHapley Additive exPlanations
SVM	Support Vector Machine

Executive Summary

The goal of the ASCAPE project is to support cancer patients' health status as much as possible and to improve their Quality of Life (QoL). The deliverable reports on the current development stage of essential ASCAPE components and services that will be fully implemented in accordance with the ASCAPE architecture until the end of the project.

The Proof of Concept (PoC) is demonstrated based on currently existing retrospective datasets from ASCAPE pilots. First part of the presented activities is focused on essential issues of the Edge-Cloud Architecture. As Kubeflow framework was chosen for the Cloud activities, k3s a lightweight production-ready Kubernetes distribution, was used for the ASCAPE Edge-Cloud Architecture PoC. Implementation of a Placeholder Container for each component in the ASCAPE Architecture is demonstrated. Additionally, two deployment configurations have been defined: one for the ASCAPE Edge Nodes and another for the ASCAPE Cloud. Further, the efforts are focused on data management and application of several powerful Machine Learning (ML) and Artificial Intelligence (AI) techniques on datasets prepared from existing retrospective data. Ten training datasets are prepared and used in experiments based on the same techniques and evaluations. The PoC contains several different stages: data pre-processing and missing values imputation, application of differential privacy methods to protect patients' personal data, the predictive federated model training including training with homomorphically encrypted data. Furthermore, for the purpose of better understanding of results obtained from models' evaluation, characteristic explainable AI methods are employed and illustrated. All the obtained results, which are extensively presented in D2.4, are satisfactory and promising in almost all aspects except for long term QoL predictions and models trained on homomorphically encrypted data as only stochastic gradient descent optimizers were used at the moment. The ASCAPE mechanisms for ensuring security and privacy of the data and integrity of the platform are of high importance. WSO2 tool which supports authentication and acts as an API gateway is selected for the ASCAPE framework. The main features of WSO2 Identity Server and API Manager are briefly presented. Kubeflow as a framework that contains a curated set of compatible tools specific to ML and running on Kubernetes (the cloud resource orchestration tool agreed with the partners) is briefly introduced. The justification of this selection as well as the key reasons are presented in detail.

1 Introduction

One of the main purposes of the ASCAPE framework is to employ powerful mechanisms of Artificial Intelligence (AI) and Machine Learning (ML) to support cancer patients' health status and Quality of Life (QoL) as much as possible. Two types of cancer are in ASCAPE's focus: breast and prostate cancer. An essential activity in ASCAPE is training predictive models for identifying QoL indicators and using them for proposing adequate, beneficial interventions for individual patients. As results of predictive models can be presented in a form that is not user friendly for physicians, several methods of explainable AI have been considered. All of them are based on the trained predictive models.

For achieving and assessing the quality of the predictive models, retrospective and prospective datasets from clinical partners will be used within the project. Until now, several retrospective datasets that are obtained from Orebro served as a basis for models training and evaluation.

To support the deployment and execution of numerous AI services and to assure the development of mechanisms safeguarding the privacy and security of the data, the complex ASCAPE architecture (presented in D1.3) consisting of numerous components will be realised. In this deliverable, implementation of several characteristic functionalities and services that cover different aspects of the ASCAPE framework, will be illustrated as initial proof-of-concept (PoC). For trustworthy functioning of the ASCAPE continuous learning, predictive analytics, and decision-making, reliable communication between ASCAPE Edge Node Platform and Cloud Platform is needed and will be supported in realization of ASCAPE architecture.

This deliverable represents the first step towards full implementation of the ASCAPE prototype. Currently, it includes PoCs for several different kinds of services that will be fully implemented and incorporated in the ASCAPE framework at the end of the project.

The deliverable is structured in accordance with the interconnected processes foreseen in WP3. It covers technical aspects of the framework that has been considered and developed until M14. Section 2 considers essential issues of the Edge-Cloud Architecture. It provides a first realization of the technical architecture based on the conceptual architecture. At the moment it presents a realization of placeholders for the main components on the Edge and on the Cloud side. Section 3 is devoted to different ASCAPE data management services. The first group of services is focused on the process of data harmonization, transformations, and aggregations from different data sources. The second group of services is oriented towards components that facilitate collection of data from wearable devices and open weather data. Section 4 is focused on ASCAPE continuous learning support. First AI based data pre-processing is implemented. The ASCAPE federated learning is illustrated by two types of predictive QoL models: global and local. At the end, the use of homomorphic encryption mechanism to train and evaluate a global model on a collection of encrypted patient data is illustrated. Section 5 gives an outline of necessary steps that should be taken within ASCAPE framework to generate explanations based on predictive models. Selected feature attribution framework is justified, and effects of interpretable surrogate models are illustrated. Mechanisms that ensure the security and privacy of the data within ASCAPE framework, i.e. a security management mechanism, an encryption block and a privacy block, are presented in Section 6. Within Section 7 evaluation and selection of a Cloud framework is performed. Thereafter, a set of proof of concepts that validate the selection is described.

While D3.1 does not provide an integrated ASCAPE Prototype following the Architecture of D1.3, but rather initial PoC work across a number of areas, it provides a good starting point for the First and Final Integrated Prototypes (D3.2 and D3.3 respectively). This can be seen in Figure 1, which presents the ASCAPE Architecture and the relevance of various sections of the present deliverable to the various ASCAPE components of the Integrated ASCAPE Prototypes.

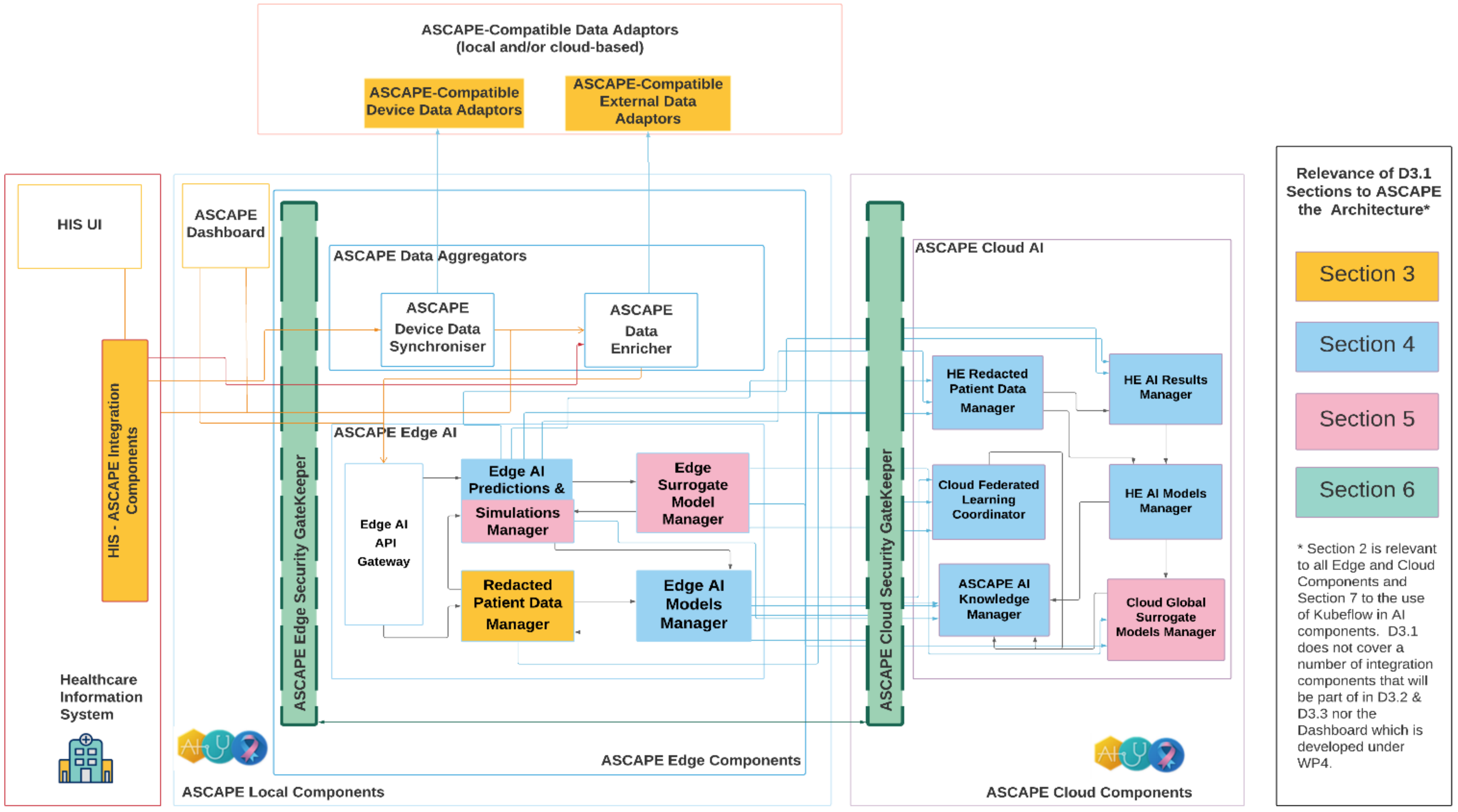

ASCAPE-Compatible Data Adaptors (local and/or cloud-based)
ASCAPE-Compatible Device Data Adaptors
ASCAPE-Compatible External Data Adaptors
HIS UI
HIS - ASCAPE Integration Components
Healthcare Information System
ASCAPE Dashboard
ASCAPE Edge Security GateKeeper
ASCAPE Data Aggregators
ASCAPE Device Data Synchroniser
ASCAPE Data Enricher
ASCAPE Edge AI
Edge AI API Gateway
Edge AI Predictions & Simulations Manager
Edge Surrogate Model Manager
Redacted Patient Data Manager
Edge AI Models Manager
ASCAPE Edge Components
ASCAPE Local Components
ASCAPE Cloud Security GateKeeper
ASCAPE Cloud AI
HE Redacted Patient Data Manager
HE AI Results Manager
Cloud Federated Learning Coordinator
HE AI Models Manager
ASCAPE AI Knowledge Manager
Cloud Global Surrogate Models Manager
ASCAPE Cloud Components
Relevance of D3.1 Sections to ASCAPE the Architecture*
Section 3
Section 4
Section 5
Section 6
* Section 2 is relevant to all Edge and Cloud Components and Section 7 to the use of Kubeflow in AI components. D3.1 does not cover a number of integration components that will be part of in D3.2 & D3.3 nor the Dashboard which is developed under WP4.

Figure 1. Relevance of work reported in D3.1 to the components of the Integrated ASCAPE Prototypes

In order to easily follow and read the document the detailed corelation between components and their descriptions in sections/subsections is given in the following table:

Table 1. Detailed correlation between components and their descriptions in sections/subsections

Sections/ASCAPE Components	2	3.2	3.3	3.4	3.5	4.1	4.2	4.3	5.1	5.2	6.1	6.2	6.3	7
HIS ASCAPE Integration Components		X	X											
ASCAPE Device Data Adapter					X									
ASCAPE External Data Adapter				X										
ASCAPE Data Enricher				X										
ASCAPE Device Data Synchronizer					X									
ASCAPE Redacted Patient Data Manager						X				X		X	X	
ASCAPE Edge AI Models Manager							X							
ASCAPE HE AI Models Manager								X						
ASCAPE Cloud Federated Learning Coordinator							X							
ASCAPE Edge AI Predictions & Simulations Manager							X		X	X				
ASCAPE AI Knowledge Manager							X		X	X				
ASCAPE HE Redacted Patient Data Manager													X	
ASCAPE HE AI Results Manager								X						X
ASCAPE Edge Surrogate Manager									X					
ASCAPE Cloud Global Surrogate Models Manager									X					
ASCAPE Edge Security Gatekeeper											X			
ASCAPE Cloud Security Gatekeeper											X			

2 Edge-Cloud Architecture Proof of Concept

The present Proof of Concept deliverable marks the beginning of the transition from requirements and a conceptual architecture to a technical architecture. It also marks the beginning of the transition from isolated proof-of-concept implementations of various key areas of the ASCAPE Framework functionality to an integrated prototype (D3.2 and D3.3).

The Edge-Cloud Architecture Proof of Concept serves the project in both of these crucial transitions as:

1. It provides a first realisation of a technical architecture based on the detailed conceptual architecture of deliverable D1.3 at the level presented in the ASCAPE Architecture Diagram (Figure 1).
2. It provides a full realisation of the said architecture albeit with placeholders for the various edge-side and the cloud-side components. These placeholders are to be replaced with actual implementations in the coming months.

The hardware requirements for the ASCAPE Framework are given in D1.1. The only update for the Proof of Concept (current deliverable) and the upcoming First and Final Integrated Prototype (D3.2 and D3.3, respectively) is that it was decided that the GPUs to be supported are CUDA-enabled nVidia GPUs, a range of GPUs available at different price points with the widest support by ML frameworks.

2.1 Edge and Cloud Orchestration Platform

Given the momentum and ubiquity of the Kubernetes platform in the Cloud space, its status as the de facto standard for microservices orchestration on the Cloud and the familiarity of technical partners with the technology, the Consortium decided on Kubernetes as the orchestration platform for the ASCAPE Cloud.

For the Edge, the Consortium considered options that would provide a solid foundation for production Edge Nodes at Healthcare Providers while utilising technology that would not require a different tool set and a different skill set than Kubernetes. It was also decided that Edge Nodes should *not* be, by default, controlled by the Cloud in terms of configuration and updates allowing Healthcare Providers to have full control over them; on the other hand, where Healthcare Providers wish to perform updates or enable automatic updates, this should be easy for their IT staff to do. Considering the above requirements, k3s stood out as the platform that could bring the world of Kubernetes onto the Edge in a production environment while also having the advantage of facilitating development for both the edge and the cloud. Unlike other lightweight Kubernetes distributions that aim to facilitate development, k3s was developed for production scenarios. It is a distribution of Kubernetes that packages all the necessary components but omits cloud provider-specific drivers and alpha-stage components in a single binary that makes it easy to create simple or even high-availability clusters on the Edge or on the Cloud with minimal resource overheads.

For the purposes of the Edge-Cloud Architecture Proof-of-Concept, k3s was used for both the Edge and the Cloud. This is in line with the aim to use k3s as a means of creating test edge and cloud environments for development purposes on development machines (e.g. laptop and desktop computers) and the aim of the Edge-Cloud Architecture Proof of Concept to be the starting point for the development of the integrated prototype of the ASCAPE Framework. As noted above, k3s is also the platform currently selected to be used in production Edge Nodes in the First and Second Prototypes. For the production version of the ASCAPE Cloud, k3s is one of the Kubernetes distributions to be considered and tested further in the upcoming months, though it should be emphasised that a choice of a different distribution for the Cloud will not affect the development of ASCAPE, nor will it mean that k3s will no longer be able to support testing both the ASCAPE Edge and the ASCAPE Cloud components on a developer's laptop as it is a fully compliant Kubernetes distribution.

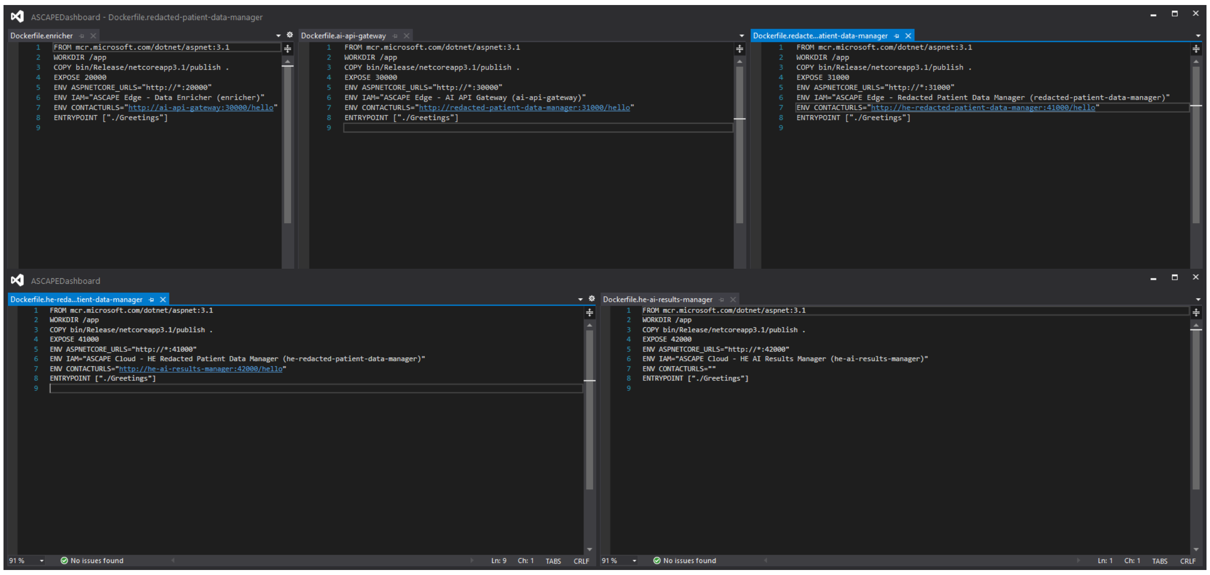
2.2 Placeholder Containers

The Edge-Cloud Proof-of-Concept Architecture aims to facilitate an implementation of the Integrated ASCAPE Prototype, starting with a Placeholder Container for each edge-side and each cloud-side component in the ASCAPE Architecture (Figure 1).

Kubernetes (and k3s) supports OCI (Open Container Initiative) container runtimes and is therefore compatible with Docker-created container images. A container image was created for each edge-based and for each cloud-based component (Table 2). The content of the container images are specified in a number of Dockerfiles (see Figure 2).

Table 2. Edge and Cloud Components in the Architecture Proof of Concept

Component	Deployment	Container Image
ASCAPE Data Enricher	Edge	Enricher
ASCAPE Device Data Synchroniser	Edge	device-data-synchroniser
Edge AI API Gateway	Edge	ai-api-gateway
Redacted Patient Data Manager	Edge	redacted-patient-data-manager
Edge AI API Predictions & Simulations Manager	Edge	predictions-simulations-manager
Edge AI Model Manager	Edge	edge-ai-model-manager
Edge Surrogate Model Manager	Edge	edge-surrogate-model-manager
HE Redacted Patient Data Manager	Cloud	he-redacted-patient-data-manager
HE AI Results Manager	Cloud	he-ai-results-manager
ASCAPE AI Knowledge Manager	Cloud	ai-knowledge-manager
Cloud Surrogate Model Manager	Cloud	cloud-surrogate-model-manager
Cloud Federated Learning Coordinator	Cloud	cloud-federated-learning-coordinator
HE AI Models Manager	Cloud	he-ai-models-manager



Dockerfile.enricher
```
FROM mcr.microsoft.com/dotnet/aspnet:3.1
WORKDIR /app
COPY bin/Release/netcoreapp3.1/publish .
EXPOSE 20000
ENV ASPNETCORE_URLS="http://*:20000"
ENV IAM="ASCAPE Edge - Data Enricher (enricher)"
ENV CONTACTURLS="http://ai-api-gateway:30000/hello"
ENTRYPOINT ["./Greetings"]
```

Dockerfile.ai-api-gateway
```
FROM mcr.microsoft.com/dotnet/aspnet:3.1
WORKDIR /app
COPY bin/Release/netcoreapp3.1/publish .
EXPOSE 30000
ENV ASPNETCORE_URLS="http://*:30000"
ENV IAM="ASCAPE Edge - AI API Gateway (ai-api-gateway)"
ENV CONTACTURLS="http://redacted-patient-data-manager:31000/hello"
ENTRYPOINT ["./Greetings"]
```

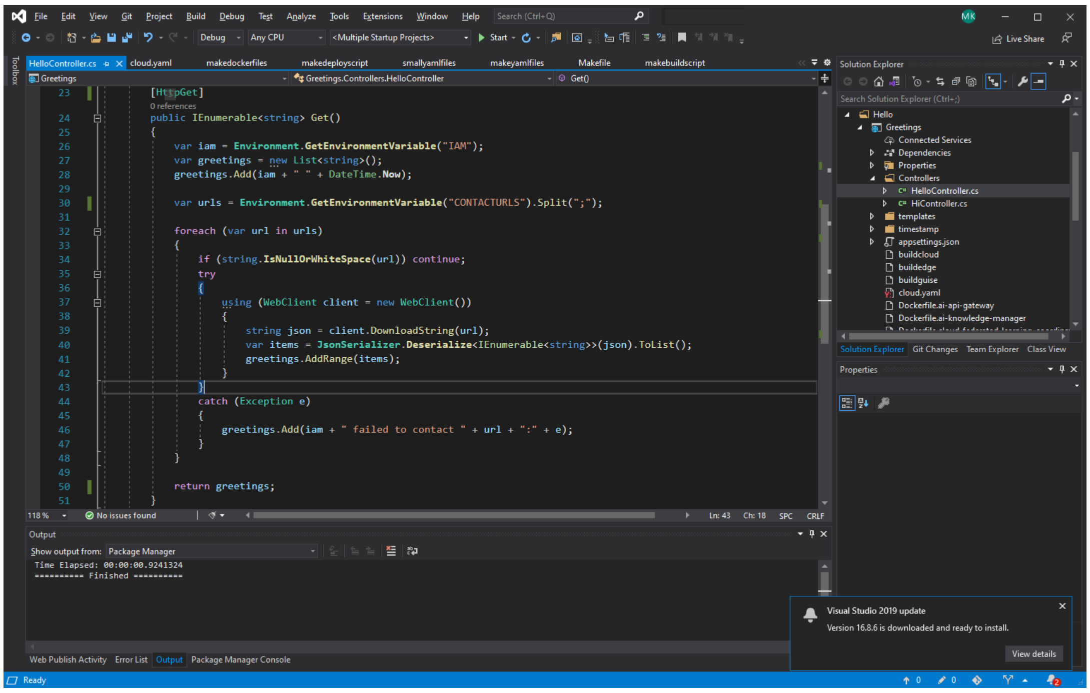
Dockerfile.redacte...atient-data-manager
```
FROM mcr.microsoft.com/dotnet/aspnet:3.1
WORKDIR /app
COPY bin/Release/netcoreapp3.1/publish .
EXPOSE 31000
ENV ASPNETCORE_URLS="http://*:31000"
ENV IAM="ASCAPE Edge - Redacted Patient Data Manager (redacted-patient-data-manager)"
ENV CONTACTURLS="http://he-redacted-patient-data-manager:41000/hello"
ENTRYPOINT ["./Greetings"]
```

Dockerfile.he-reda...tient-data-manager
```
FROM mcr.microsoft.com/dotnet/aspnet:3.1
WORKDIR /app
COPY bin/Release/netcoreapp3.1/publish .
EXPOSE 41000
ENV ASPNETCORE_URLS="http://*:41000"
ENV IAM="ASCAPE Cloud - HE Redacted Patient Data Manager (he-redacted-patient-data-manager)"
ENV CONTACTURLS="http://he-ai-results-manager:42000/hello"
ENTRYPOINT ["./Greetings"]
```

Dockerfile.he-ai-results-manager
```
FROM mcr.microsoft.com/dotnet/aspnet:3.1
WORKDIR /app
COPY bin/Release/netcoreapp3.1/publish .
EXPOSE 42000
ENV ASPNETCORE_URLS="http://*:42000"
ENV IAM="ASCAPE Cloud - HE AI Results Manager (he-ai-results-manager)"
ENV CONTACTURLS=""
ENTRYPOINT ["./Greetings"]
```

Figure 2. Dockerfiles for a selection of placeholder images corresponding to upcoming edge and cloud components

The Edge-Cloud Architecture Proof-of-Concept provides a multi-component version of the “Hello World” program capable of mimicking the interaction between ASCAPE components. The code running inside the placeholder containers (Figure 3) provides an API that returns a list of one or more greeting messages; the first is from the component contacted and the remaining, if any, from components it is configured to connect to (which may in turn return additional greetings from components they, in turn, are connected to).



HelloController.cs
```
        [HttpGet]
        0 references
        public IEnumerable<string> Get()
        {
            var iam = Environment.GetEnvironmentVariable("IAM");
            var greetings = new List<string>();
            greetings.Add(iam + " " + DateTime.Now);

            var urls = Environment.GetEnvironmentVariable("CONTACTURLS").Split(";");

            foreach (var url in urls)
            {
                if (string.IsNullOrWhiteSpace(url)) continue;
                try
                {
                    using (WebClient client = new WebClient())
                    {
                        string json = client.DownloadString(url);
                        var items = JsonSerializer.Deserialize<IEnumerable<string>>(json).ToList();
                        greetings.AddRange(items);
                    }
                }
                catch (Exception e)
                {
                    greetings.Add(iam + " failed to contact " + url + ":" + e);
                }
            }

            return greetings;
        }
```

Output
Show output from: Package Manager
```
Time Elapsed: 00:00:00.9241324
========== Finished ==========
```

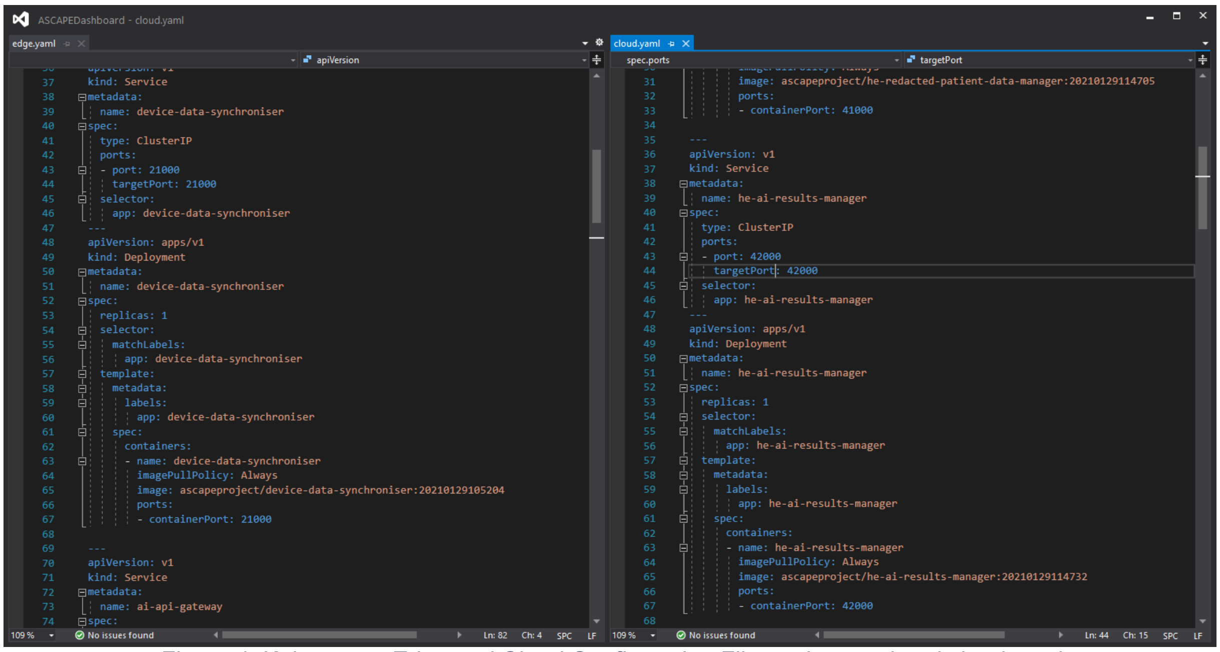
Visual Studio 2019 update
Version 16.8.6 is downloaded and ready to install.

Figure 3. Code used in Placeholder Container Images

The details of the container image definitions (Figure 2) and placeholder code (Figure 3) that result in the various placeholder containers are not important for further work in the project as each and every one of them is to be eventually replaced with an actual implementation of the corresponding component. What is important is the fact that they allow the definition of a proof-of-concept technical architecture for the Edge and the Cloud with placeholder containers corresponding to instances of the micro-services foreseen in deliverable D1.3 that are ready to be replaced with actual implementations. In addition, the Edge-Cloud Architecture Proof of Concept, as is or appropriately tweaked, can be used in preparing the hardware and network infrastructure for developing, testing and deploying ASCAPE.

2.3 Edge and Cloud Deployment

Two deployment configurations have been defined (Figure 4): one for the ASCAPE Edge Nodes (edge.yaml) and one for the ASCAPE Cloud (cloud.yaml).[1] It is possible to create a cluster with only the Edge or with only the Cloud components, or both (for integrated Edge-Cloud testing on a developer's machine). Instructions are given below.



```
kind: Service
metadata:
  name: device-data-synchroniser
spec:
  type: ClusterIP
  ports:
  - port: 21000
    targetPort: 21000
  selector:
    app: device-data-synchroniser
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: device-data-synchroniser
spec:
  replicas: 1
  selector:
    matchLabels:
      app: device-data-synchroniser
  template:
    metadata:
      labels:
        app: device-data-synchroniser
    spec:
      containers:
      - name: device-data-synchroniser
        imagePullPolicy: Always
        image: ascapeproject/device-data-synchroniser:20210129105204
        ports:
        - containerPort: 21000

---
apiVersion: v1
kind: Service
metadata:
  name: ai-api-gateway
spec:
```

```
        image: ascapeproject/he-redacted-patient-data-manager:20210129114705
        ports:
        - containerPort: 41000

---
apiVersion: v1
kind: Service
metadata:
  name: he-ai-results-manager
spec:
  type: ClusterIP
  ports:
  - port: 42000
    targetPort: 42000
  selector:
    app: he-ai-results-manager
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: he-ai-results-manager
spec:
  replicas: 1
  selector:
    matchLabels:
      app: he-ai-results-manager
  template:
    metadata:
      labels:
        app: he-ai-results-manager
    spec:
      containers:
      - name: he-ai-results-manager
        imagePullPolicy: Always
        image: ascapeproject/he-ai-results-manager:20210129114732
        ports:
        - containerPort: 42000

```

Figure 4. Kubernetes Edge and Cloud Configuration Files: edge.yaml and cloud.yaml

2.4 Installation

The ASCAPE Edge and Cloud Architecture Prototype can be deployed on a k3s cluster either in part (edge or cloud) or in whole (edge and cloud – for local deployment) using a few simple command line commands. For the instructions below, a modern Linux

[1] The edge and cloud deployment configuration files can be downloaded from https://www.dropbox.com/s/25100eb02chpfol/edge.yaml?dl=0 and https://www.dropbox.com/s/je2icg2gi5lsy62/cloud.yaml?dl=0 respectively.

distribution supported by k3s is assumed and administrator (root) privileges are required.

First, if k3s is not already installed and up-and-running, the following command will ensure it is.

```
curl -sfL https://get.k3s.io | sh -
```

Deploying the edge components, can be accomplished with the following command

```
curl -sL https://www.dropbox.com/s/25100eb02chpfol/edge.yaml?dl=0 | kubectl apply -f -
```

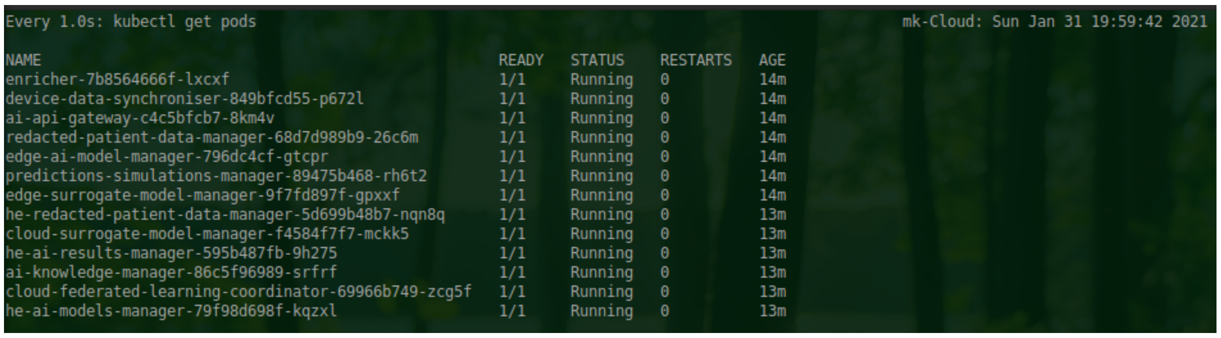
Deploying the cloud components, can be accomplished with the following command

```
curl -sL https://www.dropbox.com/s/je2icg2gi5lsy62/cloud.yaml?dl=1 | kubectl apply -f -
```

To monitor the k3s cluster at 1 second intervals, the following command can be used:

```
watch -n1 "kubectl get pods"
```

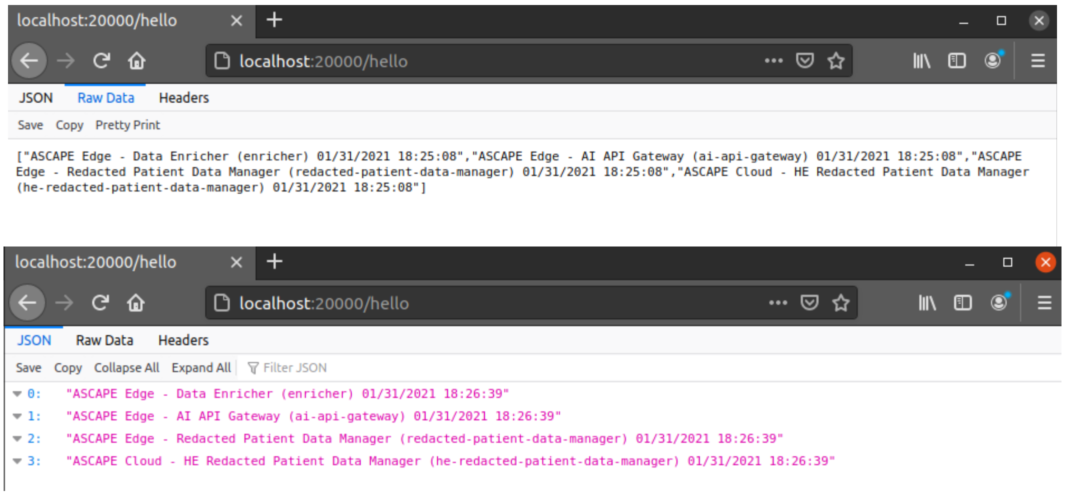
If used in a different terminal prior to deploying the edge and/or cloud components, it will offer the opportunity to follow the progress of k3s in creating the containers, downloading their images and initiating them. The expected result after the images have loaded and the containers have been initiated should be similar to what is depicted in Figure 5.



```
Every 1.0s: kubectl get pods                                          mk-Cloud: Sun Jan 31 19:59:42 2021

NAME                                                READY   STATUS    RESTARTS   AGE
enricher-7b8564666f-lxcxf                           1/1     Running   0          14m
device-data-synchroniser-849bfcd55-p672l            1/1     Running   0          14m
ai-api-gateway-c4c5bfcb7-8km4v                      1/1     Running   0          14m
redacted-patient-data-manager-68d7d989b9-26c6m      1/1     Running   0          14m
edge-ai-model-manager-796dc4cf-gtcpr                1/1     Running   0          14m
predictions-simulations-manager-89475b468-rh6t2     1/1     Running   0          14m
edge-surrogate-model-manager-9f7fd897f-gpxxf        1/1     Running   0          14m
he-redacted-patient-data-manager-5d699b48b7-nqn8q   1/1     Running   0          13m
cloud-surrogate-model-manager-f4584f7f7-mckk5       1/1     Running   0          13m
he-ai-results-manager-595b487fb-9h275               1/1     Running   0          13m
ai-knowledge-manager-86c5f96989-srfrf               1/1     Running   0          13m
cloud-federated-learning-coordinator-69966b749-zcg5f 1/1    Running   0          13m
he-ai-models-manager-79f98d698f-kqzxl               1/1     Running   0          13m
```

Figure 5. Result of monitoring ASCAPE Edge and Cloud Architecture Proof-of-Concept pods

To test that the communication between the cluster and the host machine as well as the communication pathways between the edge and cloud components are working, either a command-line tool or a browser can be used to inspect the results of the API (http://localhost:20000/hello) provided by the ASCAPE Edge-Cloud Architecture Proof of Concept.


localhost:20000/hello

JSON Raw Data Headers

Save Copy Pretty Print

```
["ASCAPE Edge - Data Enricher (enricher) 01/31/2021 18:25:08","ASCAPE Edge - AI API Gateway (ai-api-gateway) 01/31/2021 18:25:08","ASCAPE
Edge - Redacted Patient Data Manager (redacted-patient-data-manager) 01/31/2021 18:25:08","ASCAPE Cloud - HE Redacted Patient Data Manager
(he-redacted-patient-data-manager) 01/31/2021 18:25:08"]
```

localhost:20000/hello

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
0:  "ASCAPE Edge - Data Enricher (enricher) 01/31/2021 18:26:39"
1:  "ASCAPE Edge - AI API Gateway (ai-api-gateway) 01/31/2021 18:26:39"
2:  "ASCAPE Edge - Redacted Patient Data Manager (redacted-patient-data-manager) 01/31/2021 18:26:39"
3:  "ASCAPE Cloud - HE Redacted Patient Data Manager (he-redacted-patient-data-manager) 01/31/2021 18:26:39"
```

Figure 6. Using a browser to test the exposed test API

3 ASCAPE services for data management

This section is devoted to the description of different ASCAPE data management services: from data sources and collection to data transformation and visualisation. The first subsection presents data that has been processed and analysed in experiments up to now. The second subsection describes the process of data harmonisation, transformations and aggregations from different data sources, while the third subsection is dedicated to the proof of concept of the API responsible for the transformation and transmission of the new structured data into the server. Finally, the last two subsections give the overview of the proof of concept of the components that facilitate the acquisition and collection of open weather data and data from wearables in the ASCAPE framework.

3.1 Types of data

ASCAPE's main goal is to collect different types of cancer-related data from multiple sources in order to predict patients' QoL aspects, indicate clinical outcomes and be able to recommend the intervention that best suits them. Among the data sources are patient health data obtained from healthcare providers, including patient self-reported data, and other data obtained by various means like wearables or from open data sources.

For the purposes of the present Proof of Concept deliverable, anonymized retrospective breast and prostate cancer datasets and/or their description have been used for several purposes: (1) to select variables of interest and design ASCAPE common data model for data collection during pilots; (2) to create and train AI models and algorithms; and (3) to create 9 fake sample registers derived from one of Örebro's datasets that have been used in the PoC API. Detailed description of retrospective datasets from pilots can be found in deliverables D1.2 and D2.1.

The ASCAPE common data model will be followed during pilots in order to collect information in the same homogenized and standardized way for all clinical sites. By following that process, a smoother path for post analysis of patient clinical data will be achieved.

In addition to initial variables of interest, new ones will also be collected from Pilots such as sensor data, and nutritional data. Nutrition information will be collected through questionnaires and have been included into the ASCAPE data model, ready to be gathered during pilots. Using raw data or aggregate data from wearables need to be fully defined and probably not all of them will be necessary transformed into FHIR. Additional sources of data will be used to supplement clinical data and patient self-reported data. Such data aim to offer a broader perspective of factors that influence a patient's QoL and the comparative suitability of one intervention over another. Proof-of-concept work in this direction includes the initial progress towards the Fitbit wearable devices data adapter and the Weather data adapter described in more detail in the rest of the Section.

3.2 Data harmonization, transformations, aggregations, from different data sources

Regarding the retrospective datasets, to date, their transformation has been identified and the mapping to HL7 FHIR format and their codification to SNOMED CT has been performed. In spite the fact that three of the four pilot sites have provided one or several retrospective datasets with breast and/or prostate cancer data, still for the PoC we will demonstrate the transformation only with the Örebro dataset. It is worth mentioning that all the datasets are different from each other in both format and content. A homogenisation agreement has been reached, culminating in the creation of a data model common to all. Deliverable D2.1 describes the process in more depth.

Transformed data will be stored into a HAPI FHIR repository [1].The most convenient way to access and enter data in the HAPI repository fulfilling the ASCAPE common data model is through APIs. Through them, it is possible to introduce the information of interest without having knowledge of FHIR or SNOMED CT. In addition, a series of actions can be carried out, such as data retrieval. Using the API, data and information can be obtained and the final structuring and coding of the data within the model can be visualised.

3.3 API for proof of concept (test data insertion and retrieval)

Once the data model is defined and created, data must be transformed according to it and stored to the ASCAPE server. A possible way to insert not homogenised data into the server is using an API for transformation and channel the new structured data into the server (Figure 7).

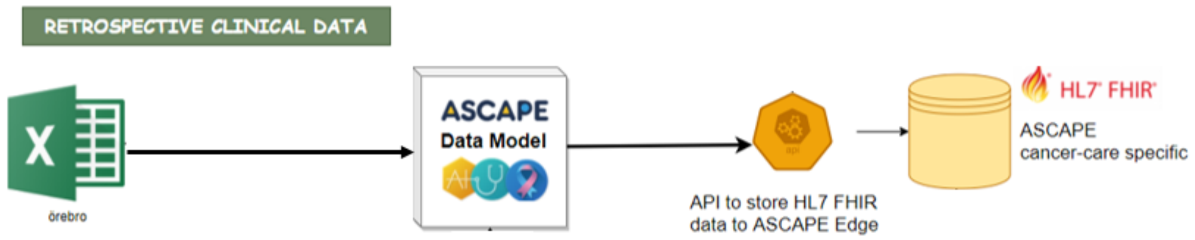

RETROSPECTIVE CLINICAL DATA
örebro
ASCAPE
Data Model
API to store HL7 FHIR data to ASCAPE Edge
HL7 FHIR
ASCAPE cancer-care specific

Figure 7. The process for transforming Örebro sample according to the ASCAPE data model and inserted into ASCAPE server through the API.

If the direction of actions is the other way (data retrieval from the server), the first contact with the structured and codified ASCAPE data model can also be made through an API (Figure 8).

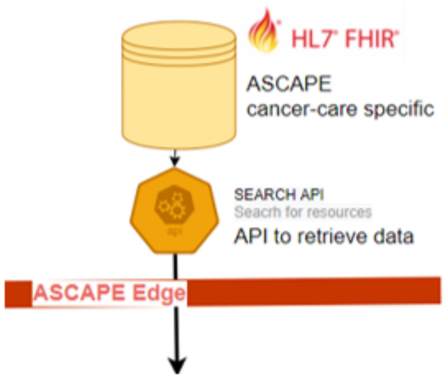
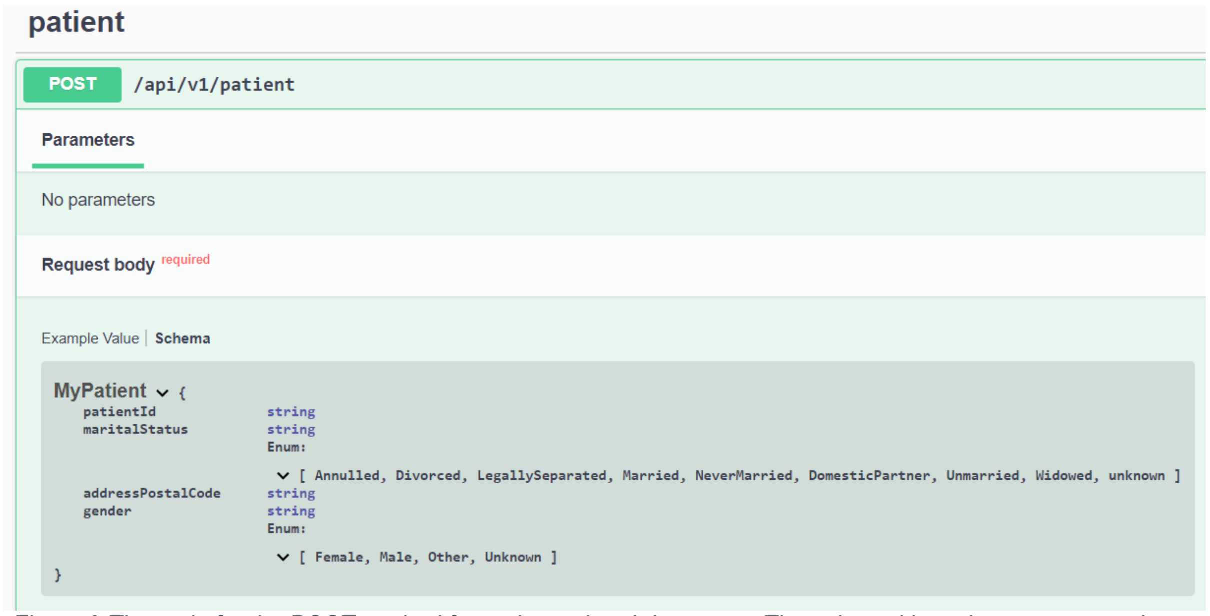


Figure 8. Scheme of how data can be retrieved from ASCAPE server through the API.

The API will assist the user in a simple and straightforward way to transform the input data into the desired common format supported by ASCAPE common data model. The API will allow the user to: (1) interact with the data in a more intuitive and human-readable way; (2) perform several tasks with the data such as collecting data from the dataset sample, adding new data and retrieving queried data; (3) comprehend and visualise how data is stored in HL7 FHIR format and codified with SNOMED CT terminology through an interface. An initial version of the API is produced as a part of the PoC and described in the following paragraphs.

The POST API (Figure 9) allows the insertion of data in a semi-automatic or automatic way, providing and indicating the fields to be completed and the way to do it (allowed values, type of variable, meaning of the content within the whole dataset, etc.).



patient

POST /api/v1/patient

Parameters

No parameters

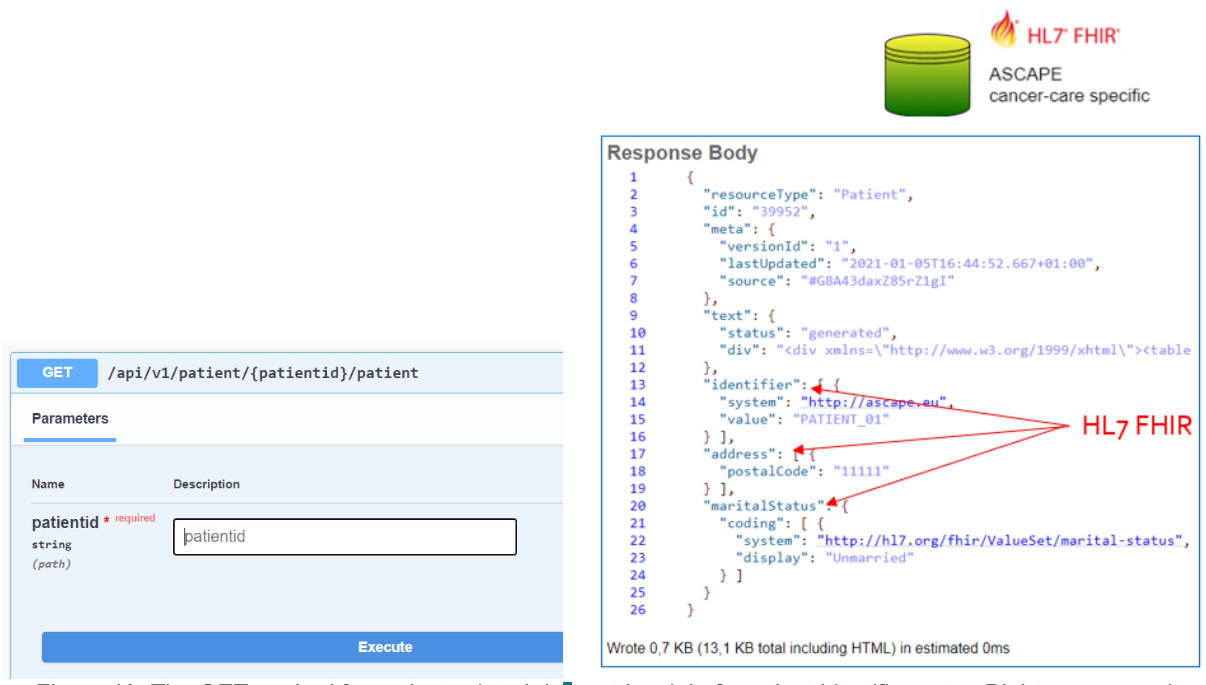
Request body required

Example Value | Schema

```
MyPatient ⌄ {
    patientId            string
    maritalStatus        string
                         Enum:
                          ⌄ [ Annulled, Divorced, LegallySeparated, Married, NeverMarried, DomesticPartner, Unmarried, Widowed, unknown ]
    addressPostalCode    string
    gender               string
                         Enum:
                          ⌄ [ Female, Male, Other, Unknown ]
}
```

Figure 9.The code for the POST method for patient-related data entry. The coloured box shows content schema (type and permitted values). They correspond to the variables of interest and match HL7 FHIR patient attributes (patient identifier, marital status, postal code and gender).

Besides, the GET API also enables the retrieval of data through simple queries, making it easier for the user to obtain information and data of interest. In Figure 10, a specific example is presented: patient information retrieval given their ID. A patient ID is represented in String format, so for the PoC simple numeric or letter values can be used as they are not real patient IDs.



```
/api/v1/patient/{patientid}/patient
```

```
{
  "resourceType": "Patient",
  "id": "39952",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2021-01-05T16:44:52.667+01:00",
    "source": "#G8A43daxZ85rZ1gI"
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><table
  },
  "identifier": [ {
    "system": "http://ascape.eu",
    "value": "PATIENT_01"
  } ],
  "address": [ {
    "postalCode": "11111"
  } ],
  "maritalStatus": {
    "coding": [ {
      "system": "http://hl7.org/fhir/ValueSet/marital-status",
      "display": "Unmarried"
    } ]
  }
}
```

Wrote 0,7 KB (13,1 KB total including HTML) in estimated 0ms

Figure 10. The GET method for patient-related data retrieval. Left: patient identifier entry. Right: query result: patient-related data structured following HL7 FHIR.

The number of variables of interest included in the PoC are less than the total amount shaping the ASCAPE common data model. Consequently, not all the variables of the ASCAPE common data model will have available content at this point. This will only be achieved after the completion of the pilots, when clinical partners collect all of them. Data used and fed into the system as a sample for this first proof of concept are 9 fake records from one of Örebro's breast cancer datasets. Fake data implies that no real patient information is used and, therefore, data is completely anonymized and patient security is not violated. The upload process has been achieved using the data flow toolApache Nifi [2], which facilitate data transformation processes and information flows between systems and files (Figure 11 and Figure 12). Apache Nifi is an ETL tool that smooths data extraction, transformation and loading for the implementer. For the PoC construction, Apache Nifi was useful to extract data from Örebro sample fake dataset, transform them into the ASCAPE common data model (HL7 FHIR-structured and Snomed CT-codified), and load transformed data into the API.

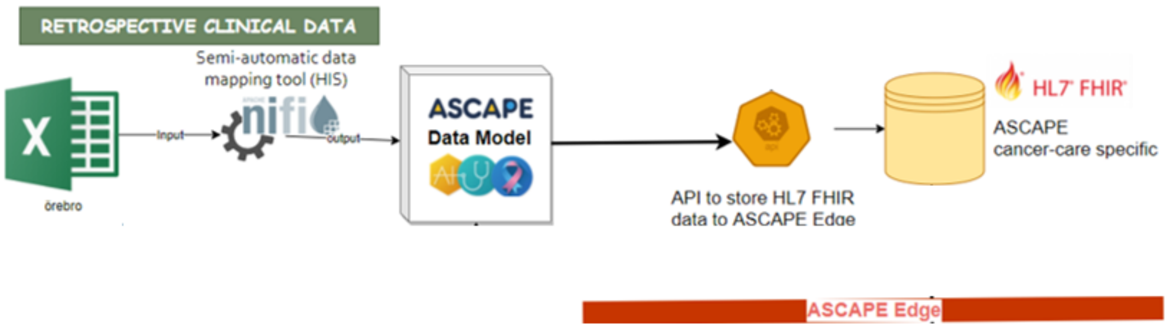
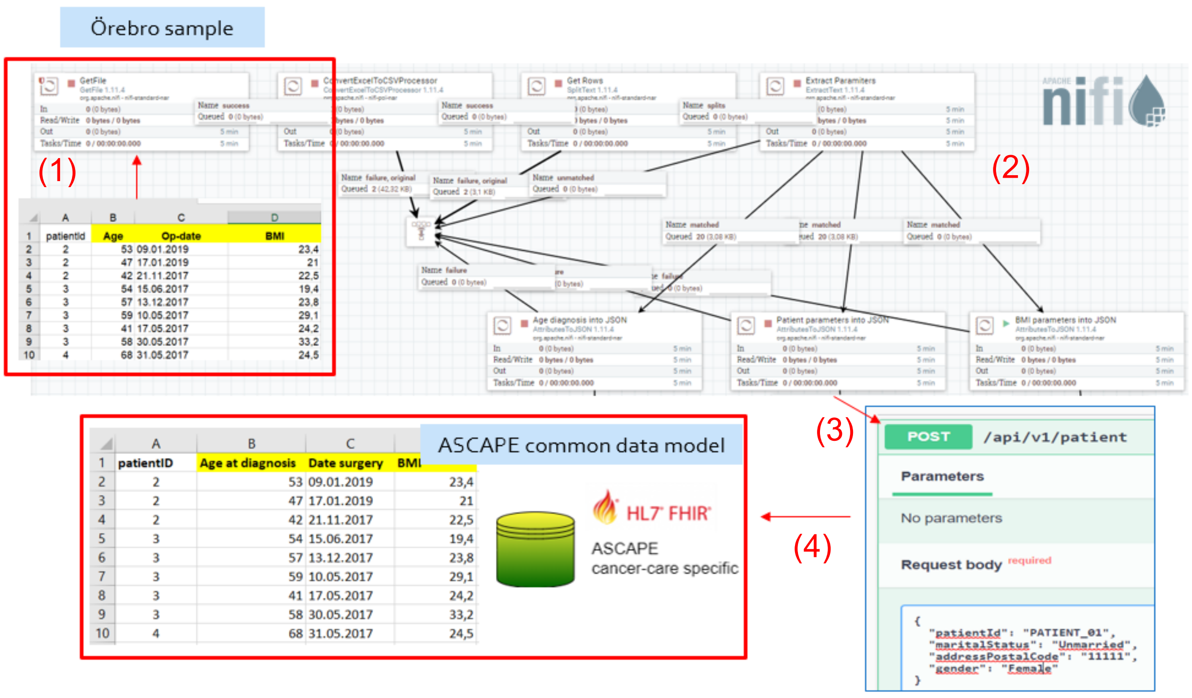

RETROSPECTIVE CLINICAL DATA
Semi-automatic data mapping tool (HIS)
örebro
input
output
ASCAPE Data Model
API to store HL7 FHIR data to ASCAPE Edge
HL7 FHIR
ASCAPE cancer-care specific
ASCAPE Edge

Figure 11. General diagram. Örebro sample automatic insertion into the API fields using Nifi.



	A	B	C	D
1	patientId	Age	Op-date	BMI
2	2	53	09.01.2019	23.4
3	2	47	17.01.2019	21
4	2	42	21.11.2017	22.5
5	3	54	15.06.2017	19.4
6	3	57	13.12.2017	23.8
7	3	59	10.05.2017	29.1
8	3	41	17.05.2017	24.2
9	3	58	30.05.2017	33.2
10	4	68	31.05.2017	24.5

	A	B	C	
1	patientID	Age at diagnosis	Date surgery	BMI
2	2	53	09.01.2019	23,4
3	2	47	17.01.2019	21
4	2	42	21.11.2017	22,5
5	3	54	15.06.2017	19,4
6	3	57	13.12.2017	23,8
7	3	59	10.05.2017	29,1
8	3	41	17.05.2017	24,2
9	3	58	30.05.2017	33,2
10	4	68	31.05.2017	24,5

POST /api/v1/patient

Parameters

No parameters

Request body required

```
{
  "patientId": "PATIENT_01",
  "maritalStatus": "Unmarried",
  "addressPostalCode": "11111",
  "gender": "Female"
}
```

Figure 12. Process: (1) sample data caption using Apache Nifi; (2) Nifi extraction of information matching API schema; (3) API fields are filled and data transformed; (4) homogenised and structured data is stored in ASCAPE server following ASCAPE common data model, HL7 FHIR and SNOMED CT.

Amongst the information contained in the sample dataset and dumped into the ASCAPE system through the API for the PoC are the following:

1) patient-related data such as identifier, BMI, age at diagnosis;

2) tumour-related data such as diameter, body site/laterality, histological type, grade, N stage and T stage;

3) performed treatment such as type and date of surgery, type of endocrine therapy and postoperative complications;

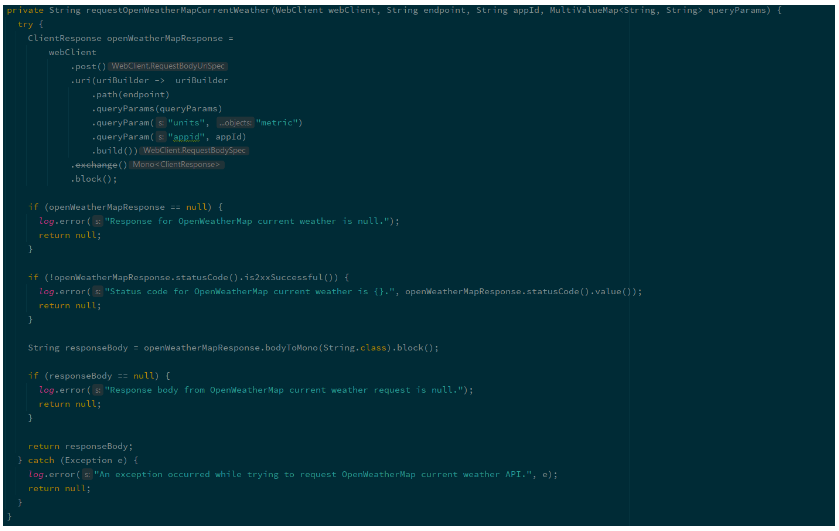
4) other relevant medical conditions such as Charlson Comorbidity Index, percentage of estrogen and progesterone receptors, HER2 positivity, Ki-67 factor and number of positive lymph and sentinel nodes; and

5) quality of life and satisfaction questionnaires.

3.4 The ASCAPE Weather Adapter

The **ASCAPE Weather Data Adapter** is an External-Data Adapter offered by the ASCAPE framework to facilitate the acquisition and collection of weather open-data originating from a predefined list of open weather data sources. The adapter's implementation is following a microservices-based architecture and it is composed of a set of microservices, each one implemented with a specific context and role in the adapter's designed solution. The designed solution is aligned with the HIS Data Synchronization and Enrichment process, as defined in deliverable D1.3 and constitutes the proof of concept for the ASCAPE Weather Data Adapter in M14 towards the fully operational version of the adapter that is expected on M24.

The first microservice named Weather Data Handler is providing the periodic schedulers of the adapter for each weather open-data source, based on the data source profiles and an intermediate storage. The second microservice named Orchestrator is undertaking the intercommunication between the Weather Data Handler and the third microservice named Weather Data Parser, which is being invoked by the Data Handler, once a scheduler is successfully executed. The third microservice, namely the Weather Data Parser, undertakes the task of retrieving the new information from the intermediate storage, parsing and indexing the new information. The fourth microservice named Weather Data Explorer is providing the ASCAPE-specific API of the ASCAPE Weather Data Adapter which is leveraged by the Data Enricher component in order to retrieve the required weather information for a specific area and a specific period.



```
private String requestOpenWeatherMapCurrentWeather(WebClient webClient, String endpoint, String appId, MultiValueMap<String, String> queryParams) {
  try {
    ClientResponse openWeatherMapResponse =
        webClient
            .post() WebClient.RequestBodyUriSpec
            .uri(uriBuilder ->  uriBuilder
                .path(endpoint)
                .queryParams(queryParams)
                .queryParam( s: "units",  ...objects: "metric")
                .queryParam( s: "appid", appId)
                .build()) WebClient.RequestBodySpec
            .~~exchange~~() Mono<ClientResponse>
            .block();

    if (openWeatherMapResponse == null) {
      log.error( s: "Response for OpenWeatherMap current weather is null.");
      return null;
    }

    if (!openWeatherMapResponse.statusCode().is2xxSuccessful()) {
      log.error( s: "Status code for OpenWeatherMap current weather is {}.", openWeatherMapResponse.statusCode().value());
      return null;
    }

    String responseBody = openWeatherMapResponse.bodyToMono(String.class).block();

    if (responseBody == null) {
      log.error( s: "Response body from OpenWeatherMap current weather request is null.");
      return null;
    }

    return responseBody;
  } catch (Exception e) {
    log.error( s: "An exception occurred while trying to request OpenWeatherMap current weather API.", e);
    return null;
  }
}
```

Figure 13. ASCAPE Weather Data Adapter - The periodic scheduler of Data Handler

The implementation of the ASCAPE Weather Data Adapter is currently on the following private Gitlab repositories and access can be given upon request:

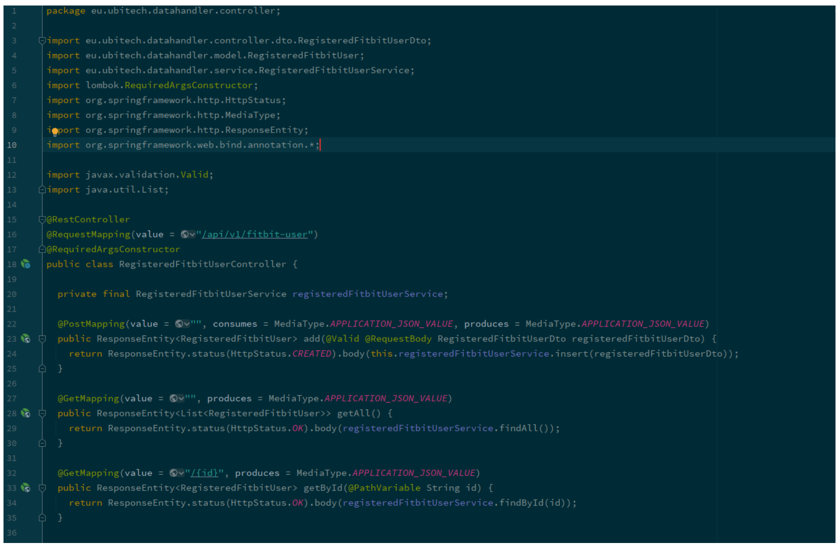
- https://gitlab.ubitech.eu/dsa/ascape/data-workflow-orchestrator
- https://gitlab.ubitech.eu/dsa/ascape/data-handler

- https://gitlab.ubitech.eu/dsa/ascape/data-parser
- https://gitlab.ubitech.eu/dsa/ascape/data-explorer

3.5 The Fitbit Device Data Adapter

The **Fitbit Device Data Adapter** is an ASCAPE Device Data Adapter offered by the ASCAPE framework to enable the retrieval and local storage of data originating from the family of Fitbit wearables. For the implementation of the specific adapter the microservices pattern is also adopted. The Fitbit Device Data Adapter is composed by four core microservices, namely the Adapter Controller, the Fitbit Data Scheduler, the Fitbit Data Indexer and the Fitbit Data Retriever.

The role of the Adapter Controller is to facilitate the execution of the overall adapter process, from the data collection from the Fitbit Cloud service to the retrieval of the locally collected data from the ASCAPE Device Data Synchronizer, as described in the Device Data Synchronization process in deliverable D1.3. The Fitbit Data Scheduler undertakes the core functionality of periodically interacting with the Fitbit Web API in order to fetch the data of the monitored Fitbit devices, as instructed by the ASCAPE Device Data Synchronizer. The Fitbit Data Indexer is responsible for the interpretation and indexing of the locally downloaded data from the Fitbit devices. Finally, the Fitbit Data Retriever undertakes the implementation of the ASCAPE-specific API of the Fitbit Device Data Adapter that serves the data retrieval requests originating from the ASCAPE Device Data Synchronizer.



```java
package eu.ubitech.datahandler.controller;

import eu.ubitech.datahandler.controller.dto.RegisteredFitbitUserDto;
import eu.ubitech.datahandler.model.RegisteredFitbitUser;
import eu.ubitech.datahandler.service.RegisteredFitbitUserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping(value = "/api/v1/fitbit-user")
@RequiredArgsConstructor
public class RegisteredFitbitUserController {

  private final RegisteredFitbitUserService registeredFitbitUserService;

  @PostMapping(value = "", consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<RegisteredFitbitUser> add(@Valid @RequestBody RegisteredFitbitUserDto registeredFitbitUserDto) {
    return ResponseEntity.status(HttpStatus.CREATED).body(this.registeredFitbitUserService.insert(registeredFitbitUserDto));
  }

  @GetMapping(value = "", produces = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<List<RegisteredFitbitUser>> getAll() {
    return ResponseEntity.status(HttpStatus.OK).body(registeredFitbitUserService.findAll());
  }

  @GetMapping(value = "/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<RegisteredFitbitUser> getById(@PathVariable String id) {
    return ResponseEntity.status(HttpStatus.OK).body(registeredFitbitUserService.findById(id));
  }

```

Figure 14. ASCAPE Fitbit Device Data Adapter - The periodic scheduler of Fitbit Data Scheduler

The described solution is provided as a proof of concept in M14 and it will drive the implementation activities towards the fully operational implementation of the adapter that will be delivered on M24. The implementation of the Fitbit Device Data Adapter is currently on the following private Gitlab repositories and access can be give upon request:

- https://gitlab.ubitech.eu/dsa/ascape/data-workflow-orchestrator

- https://gitlab.ubitech.eu/dsa/ascape/data-handler
- https://gitlab.ubitech.eu/dsa/ascape/data-parser
- https://gitlab.ubitech.eu/dsa/ascape/data-explorer

3.6 Implementation technologies for PoC demonstration

The main component of the PoC for data transformation is the API through which users can interact (insert and retrieve data). The API was implemented using Swagger UI [3] and Eclipse IDE. Data format in the API follows ASCAPE common data model which is HL7 FHIR-structured and Snomed CT-codified. The creation of this common data model was developed using Java and Eclipse IDE, incorporating FHIR dependencies. Fake data registers for PoC were extracted from the original file and loaded into the API using Apache Nifi ETL tool.

The implementation of both the ASCAPE Weather Data Adapter and the FitBit Device Data Adapter is based on a set of well-established open-source libraries and frameworks. The adapters are implemented in Java 11 where the Spring Boot framework [4] is utilised. Beyond this framework, MongoDB [5] is utilised for the data source profile management as well as the adapters' operational storage, MinIO [6] storage server is utilised as an intermediate storage and Elasticsearch [7] is exploited as an indexing engine. Finally, in order to fully leverage the query capabilities of Elasticsearch, GraphQL Spring Boot library [8] is employed on top of it.

4 ASCAPE Continuous learning support

In this section we will give an overview of the continuous learning support within ASCAPE project. First, we give implementational details regarding dataset pre-processing steps. Then we present all the modules that support federated learning. Finally, we describe how homomorphic encryption is implemented within the project.

4.1 AI based data pre-processing

Before training AI models, the available datasets must be pre-processed. Some pre-processing steps are mandatory, while the others only improve the predictive power of trained models. In the following text we will present and demonstrate three AI based pre-processing steps: missing values inference (MVI), outlier detection and removal, and Differential Privacy (DP).

4.1.1 Missing values inference (MVI)

Many AI models are incapable of processing data with missing values. Therefore, before training such models, missing values must be imputed. The missing values imputation is implemented in the module `mvi`. Before imputing missing values, the instances without target attribute are removed. After that, a missing value inference algorithm is applied on the input dataset. As a final step, a new column named “fold” is added in the dataset, so that each dataset instance gets its fold value. This step is needed for the experiments’ validation and will be thoroughly explained in the following text. Finally, the resulting dataset is output to the desired path.

The module `mvi` offers two different MVI algorithms: simple imputer and iterative imputer. Simple imputer fills missing values within a single attribute by using the mean of the attribute’s existing values. Iterative imputer is a bit more complex: it creates a regression model for each attribute and then it fills missing values of an attribute with predictions obtained from the attribute’s regression model. A regression model for a single attribute is fit on instances with non-empty value for that attribute. After filling all the missing values, iterative imputer repeats the whole procedure predefined number of times. For both MVI algorithms, we used corresponding scikit-learn [9] implementations(`sklearn.impute.SimpleImputer`, `sklearn.impute.IterativeImputer`).

After the MVI algorithm is performed, stratified and randomized 10-fold division is performed on the dataset’s instances set. The fold values are placed in the aforementioned new dataset column named “fold”. By having this column, different experiments can perform 10-fold cross validation upon identical folds, which makes the results of such experiments fully comparable. For the 10-fold division we used scikit-learn implementation `sklearn.model_selection.StratifiedKFold`.

The module `mvi` is rather simple – it has a function `apply_mvi` that takes the path to the input set, the path to the output file, and the algorithm that will be used for missing values imputation (“simple” for simple imputer, “iterative” for iterative imputer). The function performs all the aforementioned steps. An example of `apply_mvi` function call is given in Figure 15.

```
from mvi import mvi
mvi.apply_mvi('BcBase-Anxiety.csv', 'BcBase-Anxiety-Processed.csv', 'iterative')
```

Figure 15. A Python program that applies missing values inference on BcBase-Anxiety dataset.

The module `mvi` also provides a CLI. The arguments provided by the CLI are given in Figure 16.

```
 -d, --dataset
     Path to the dataset.
 -o, --out (optional)
     Path to the output file. If not given, resulting dataset will be saved
 in the current directory.
 -a, --algorithm (optional)
     MVI algorithm. Possible values:
     simple - takes the mean of attribute's values, and uses it to fill
 the missing values,
     iterative (default) - creates a predictive model for each attribute,
 and uses it to fill the missing values.
 -h, --help (optional)
     Displays help.
```

Figure 16. CLI arguments of the module mvi.

4.1.2 Outlier detection and removal

The second pre-processing step is outlier detection and removal. Outliers are dataset instances that differ too much from the other instances. Outliers can negatively influence performance of AI models, and should sometimes be removed prior to AI model training.

A mechanism for outlier detection and removal is implemented in the module `outliers`. In this module we used autoencoders for outlier detection: an autoencoder is trained on the whole dataset, and then the instances with the highest reconstruction losses are removed from the dataset. For the encoder we used a neural network with three dense layers that have 32, 16 and 8 output units, respectively, and that use relu activation function. For the decoder we used a neural network with three dense layers that have 16, 32 and *F* output units, *F* being the number of dataset features. For the first two decoder layers we used relu activation function, while for the last one we used sigmoid activation function. For this part of the project we used keras library [10].

After feeding the mentioned autoencoder with all dataset instances, we calculate the median and the standard deviation of all instances' loss values. Then we set the outlier threshold by using the following formula:

$$threshold = m + t_{mul}\sigma$$

m being median of loss values, σ being standard deviation, and t_{mul} being user-defined threshold multiplier which defaults to 1. Each instance that has reconstruction loss higher than $threshold$ is removed from the dataset.

The module `outliers` provides a function `remove_outliers` that takes the path to the input set, the path to the output file, and the threshold multiplier. The function applies the outlier removal algorithm and stores the resulting dataset to the desired path. An example of `remove_outliers` function call is given in Figure 17.

```
from outliers import outliers
outliers.remove_outliers('BcBase-Anxiety.csv', 'BcBase-Anxiety-Processed.csv', 2)
```

Figure 17. A Python program that applies outliers removal algorithm on BcBase-Anxiety dataset.

The module `outliers` also provides a CLI. The arguments provided by the CLI are given in Figure 18.

```
-d, --dataset
    Path to the dataset.
-o, --out
    Path to the output file.
-t, --threshold (optional)
    Outlier threshold multiplier.
-h, --help (optional)
    Displays help.
```

Figure 18. CLI arguments of the module outliers.

4.1.3 Differential privacy

The third pre-processing step is differential privacy. The task of this step is to protect the ML models from model inversion attacks. By performing these attacks, an attacker can reveal some amount of private information from the participants which participated in the training set just by querying the model. In order to prevent these kinds of attacks, a controlled amount of noise is added to the training data. This noise addition is performed on previously prepared data, with imputed missing values and with already removed outliers. DP is a pre-processing task but it is also a part of ASCAPE Security and privacy toolkit, so a more detailed explanation will be given in Section 6.

4.2 Federated learning

The ASCAPE federated learning is related to training (creating and updating) of two types of predictive QoL models: global models (models collectively trained on ASCAPE edge nodes without patient data exchange between nodes) and local models (models trained considering only training data available at a particular ASCAPE edge node). Predictions made by global models are derived from knowledge captured from various ASCAPE edge nodes. Local models reflect specificities of data at individual edge nodes and enable accurate predictions for specialized and rarely represented cohorts of patients. An ASCAPE edge node uses a local model for making QoL related predictions instead of the corresponding global model if the global model exhibits a low accuracy (or a high error) on training data available on that node. In this Section we present the current implementation of core ASCAPE machine learning services enabling the continuous learning of global and local models. Additionally, we demonstrate the first fully functional prototypes of the federated learning server and clients.

The core ASCAPE edge node machine learning services are enabled by a set of Python modules implemented using Scikit-learn [9] and Tensorflow [11] machine learning libraries. Those modules define classes and functions for training, storing, evaluating and applying both classification models (models predicting a discrete variable) and regression models (models predicting a numeric variable). The ASCAPE machine learning core contains the following modules:

- `loader` – a module for loading datasets present at ASCAPE edge nodes,
- `skcl` – a module for training and storing classification models using classifier learning algorithms provided by Scikit-learn,
- `skreg` – a module for training and storing regression models using regression learning algorithms provided by Scikit-learn,
- `skie` – a module realizing an inference engine (i.e., engine for making predictions) based on previously learned Scikit-learn based regression and classification models,
- `tfnn` – a module for training and storing machine learning models based on Tensorflow neural networks performing either classification or regression,
- `tfnnie` – a module providing an inference engine for Tensorflow based classification and regression models, and
- `evalm` – a module providing various evaluation measures for classification and regression machine learning models.

The `loader` module defines classes `CSVDataset` and `FoldedDataset`. Both classes load a csv dataset assuming that it does not contain missing values, i.e. they accept datasets previously treated by the missing value inference module described in Section 4.1. The `FoldedDataset` class loads csv datasets containing a special field determining how data instances are divided into separate folds when performing the K-fold cross validation. `CSVDataset` is used to load csv datasets without explicit fold assignments. `CSVDataset` treats the last column as the target or outcome variable (variable for which we want to build a predictive model), while all previous columns correspond to predictors variables (input variables used by the model to derive the value of the outcome variable). `FoldedDataset` is based on the assumption that the last two columns correspond to the target variable and fold assignment, respectively, while all previous columns are predictor variables. Both contains a method for separating the loaded dataset into two numpy arrays, one multidimensional containing predictor variables and one unidimensional containing the target variable. `FoldedDataset` additionally defines methods for returning the training and test dataset for a specified fold.

4.2.1 Local models

The `skcl` module contains a base class for training local Scikit-learn based classification models called `SKLCModel`. This class defines two generic methods: one for training the model and another for storing the model as pickled Python objects. The following classes are derived from `SKLCModel`:

- `SVM_SKLC` – a class for training support vector machine classifiers,
- `RF_SKLC` – a class for training random forest classifiers,
- `NB_SKLC` – a class for training Naïve Bayes classifiers,
- `KNN_SKLC` – a class for training K-nearest neighbors classifiers, and
- `DT_SKLC` – a class for training decision tree classifiers.

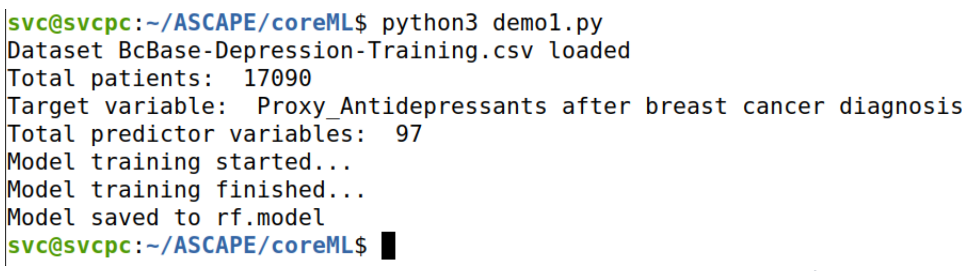
A base class for training Scikit-learn based regression models, called `SKLRModel`, is defined in the `skreg` module. This class defines generic methods for training and storing regression models. The derived classes are:

- `DummyReg` – a dummy regressor always predicting the average value of the outcome variable from the training dataset,
- `LinearReg_SKLR` – linear regression,
- `RidgeReg_SKLR` – ridge regression,
- `LassoReg_SKLR` – lasso regression,
- `ElasticNetReg_SKLR` – elastic net regression,
- `KernelRidgeReg_SKLR` – kernel ridge regression,
- `SVM_SKLR` – support vector machine regression,
- `RF_SKLR` – random forest regression,
- `KNN_SKLR` – K-nearest neighbors regression, and
- `AdaBoost_SKLR` – AdaBoost regression.

The `skie` module defines a class called `SKModel` for making predictions by previously trained and stored Scikit-learn based classification and regression models. The constructor of `SKModel` has one parameter which is the path to a stored model. The class defines two methods: `predictSingle` for making prediction for exactly one data instance (one patient) and `predictMultiple` for making predictions for multiple data instances (a set of patients).

Python demo programs illustrating how to use modules realizing the ASCAPE edge node core ML services to train, save, load and make predictions using a classification model on patients from the BcBase dataset are shown in Figure 19 and Figure 20 Figure 21 and Figure 22 show equivalent demos for a regression model on patients from the Orebro dataset.

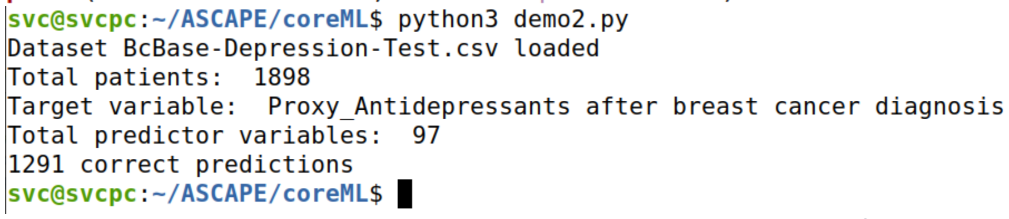
```python
from loader import CSVDataset
from skcl import RF_SKLC

# load training dataset
dataset = CSVDataset("BcBase-Depression-Training.csv")
predictors, target = dataset.getData()

# train a model (random forest; can be any other model from skcl)
model = RF_SKLC()
model.train(predictors, target)

# save the model
model.save("rf.model")
```



```
svc@svcpc:~/ASCAPE/coreML$ python3 demo1.py
Dataset BcBase-Depression-Training.csv loaded
Total patients:  17090
Target variable:  Proxy_Antidepressants after breast cancer diagnosis
Total predictor variables:  97
Model training started...
Model training finished...
Model saved to rf.model
svc@svcpc:~/ASCAPE/coreML$
```

Figure 19. A Python program demonstrating how to train and save a classification model trained on a part of the BcBase dataset using modules realizing the ASCAPE edge node core ML services.

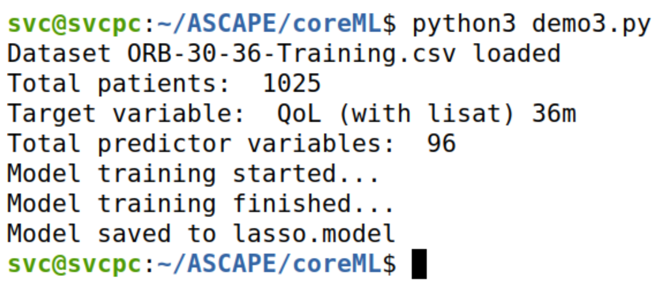
```
from loader import CSVDataset
from skie import SKModel

# load the previously created model
model = SKModel("rf.model")

# load test data
dataset = CSVDataset("BcBase-Depression-Test.csv")
predictors, target = dataset.getData()

# make inference on test data
correctPredictions = 0
for i in range(0, len(predictors)):
    prediction = model.predictSingle(predictors[i])
    if prediction == target[i]:
        correctPredictions += 1

print(correctPredictions, "correct predictions")
```



```
svc@svcpc:~/ASCAPE/coreML$ python3 demo2.py
Dataset BcBase-Depression-Test.csv loaded
Total patients:  1898
Target variable:  Proxy_Antidepressants after breast cancer diagnosis
Total predictor variables:  97
1291 correct predictions
svc@svcpc:~/ASCAPE/coreML$
```

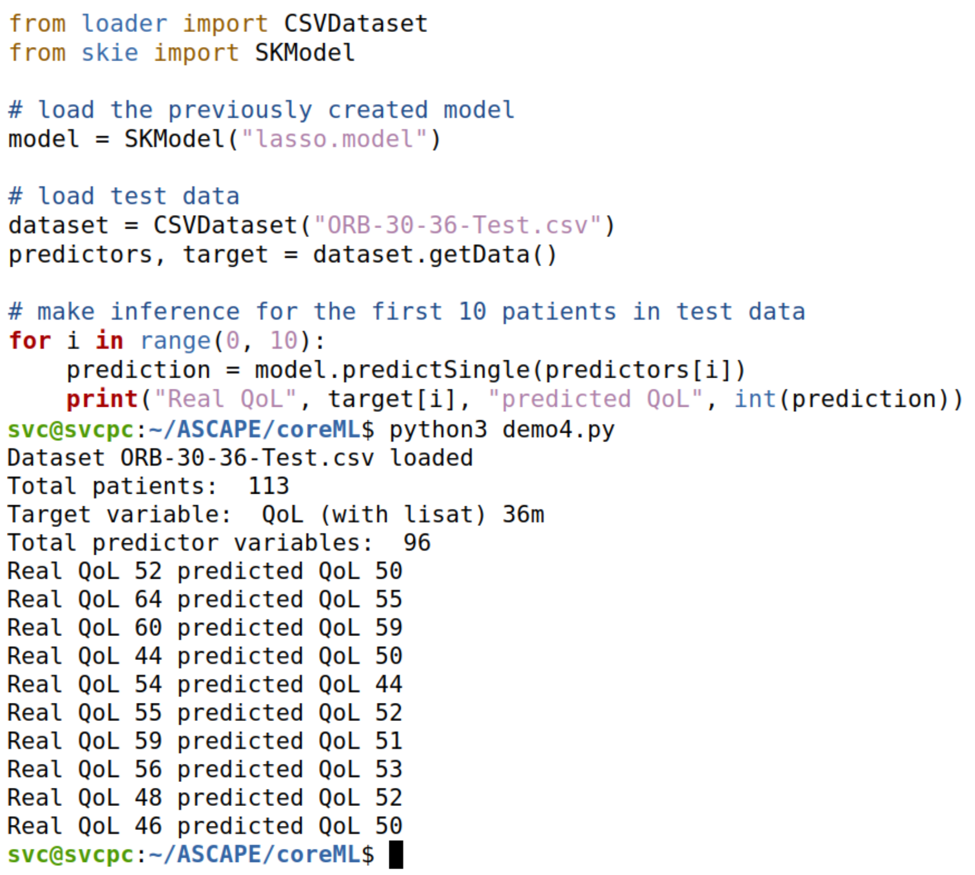
Figure 20. A Python program illustrating how to load previously trained classification model and to use it for making predictions on a part of the BcBase dataset (that was not used to train the model) using modules realizing the ASCAPE edge node core ML services.

```
from loader import CSVDataset
from skreg import LassoReg_SKLR

# load training dataset
dataset = CSVDataset("ORB-30-36-Training.csv")
predictors, target = dataset.getData()

# train a model (lasso regressor; can be any other model from skreg)
model = LassoReg_SKLR()
model.train(predictors, target)

# save the model
model.save("lasso.model")
```



```
svc@svcpc:~/ASCAPE/coreML$ python3 demo3.py
Dataset ORB-30-36-Training.csv loaded
Total patients:  1025
Target variable:  QoL (with lisat) 36m
Total predictor variables:  96
Model training started...
Model training finished...
Model saved to lasso.model
svc@svcpc:~/ASCAPE/coreML$
```

Figure 21. A Python program demonstrating how to train and save a regression model trained on a part of the Orebro dataset using modules realizing the ASCAPE edge node core ML services.



```
from loader import CSVDataset
from skie import SKModel

# load the previously created model
model = SKModel("lasso.model")

# load test data
dataset = CSVDataset("ORB-30-36-Test.csv")
predictors, target = dataset.getData()

# make inference for the first 10 patients in test data
for i in range(0, 10):
    prediction = model.predictSingle(predictors[i])
    print("Real QoL", target[i], "predicted QoL", int(prediction))
```

```
svc@svcpc:~/ASCAPE/coreML$ python3 demo4.py
Dataset ORB-30-36-Test.csv loaded
Total patients:  113
Target variable:  QoL (with lisat) 36m
Total predictor variables:  96
Real QoL 52 predicted QoL 50
Real QoL 64 predicted QoL 55
Real QoL 60 predicted QoL 59
Real QoL 44 predicted QoL 50
Real QoL 54 predicted QoL 44
Real QoL 55 predicted QoL 52
Real QoL 59 predicted QoL 51
Real QoL 56 predicted QoL 53
Real QoL 48 predicted QoL 52
Real QoL 46 predicted QoL 50
svc@svcpc:~/ASCAPE/coreML$
```

Figure 22. A Python program showing how to load previously trained regression model and to use it for making predictions on a part of the Orebro dataset (that was not used to train the model) using modules realizing the ASCAPE edge node core ML services.

4.2.2 Global models

Neural networks are the default machine learning model for global predictive models collectively trained on ASCAPE edge nodes. The `tfnn` module contains an abstract base class for models based on Tensorflow (deep) neural networks. This class, called `FedTFNNModel`, defines the following concrete (non-abstract) methods:

- `createModelDefinition` – to specify the architecture of the neural network, the loss function, the activation function and regularization mechanisms,

- `initModelStructure` – to create the model structure (neural network layers and their ordering) according to the previously given specification,
- `setupTrainingData` – to specify training data instances,
- `update` – to update the model on previously prepared training data for given number of epochs, batch size, class weights (optional parameter) and callbacks (optional parameter; callbacks are functions executed after each epoch),
- `oneLearningRound` – to perform model update in exactly one epoch,
- `setWeigths` – to set weights of neural network edges,
- `getWeights` – to obtain neural network edges, and
- `saveModel` – to store the model.

`FedTFNNModel` additionally defines two abstract methods:

- `initLastLayer` – to create the last layer in the neural network (regression and classification neural networks differ in the last layer), and
- `prepareTrainigData` – to prepare training data instance for a particular neural network type (in case of classification neural networks values of the target variable should be transformed one-hot encoding vectors, which is not the case for regression neural networks).

Three classes are derived from `FedTFNNModel` providing specific implementations of its abstract methods:

- `FedTFNNRegModel` – a Tensorflow neural network preforming regression,
- `FedTFNNClModel` – a Tensorflow neural network preforming *n*-ary classification, and
- `FedTFNNBinClModel` – a Tensorflow neural network performing binary classification.

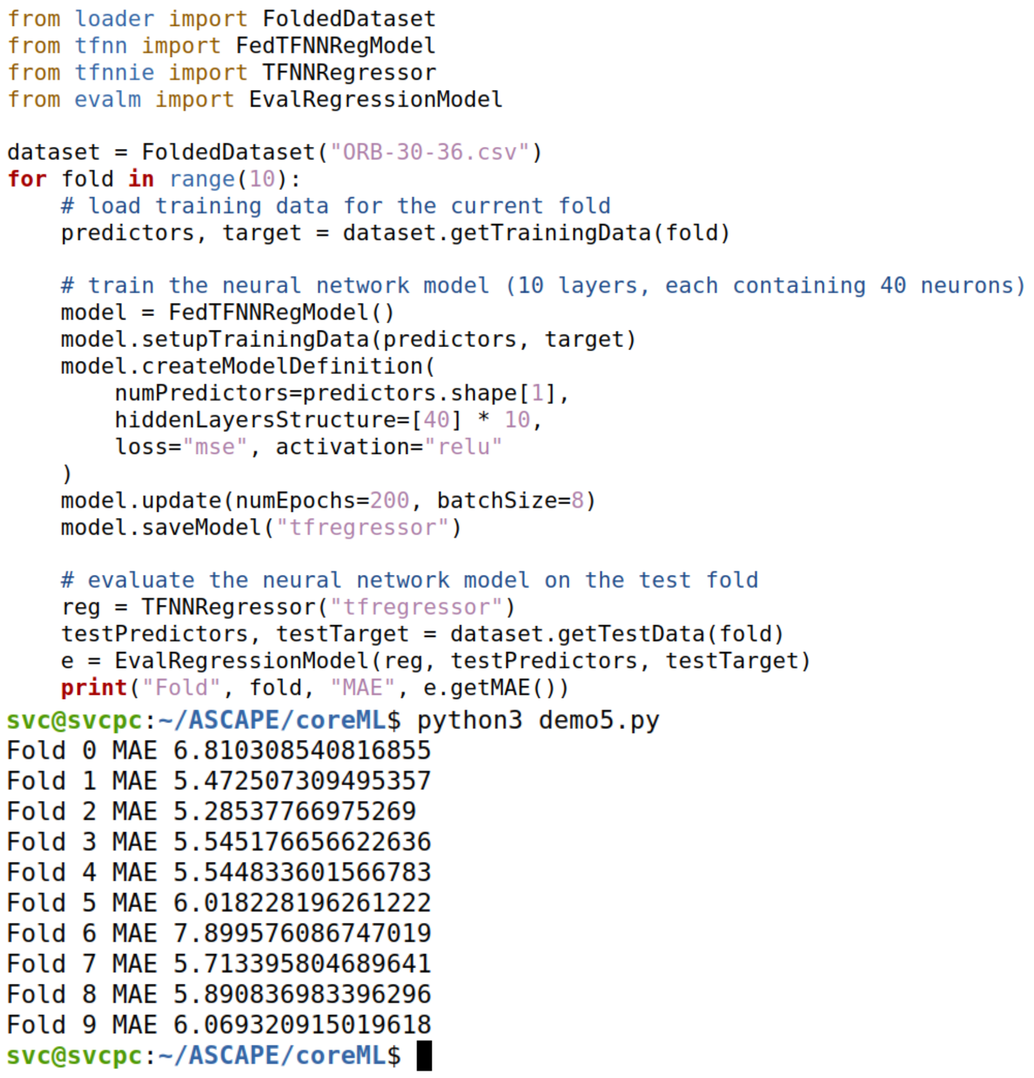
The `tfnnie` module provides inference engines for models based on Tensorflow neural networks. This module contains an abstract class `TFNNInference`. This class realizes the loading of a stored Tensorflow neural network model and it defines two abstract methods for making predictions based on the loaded model: `predictSingle` for making predictions on a single data instance and `predictMultiple` for making predictions on multiple data instances. Three classes are derived from this base class providing specific implementations of its abstract methods for different neural network types:

- `TFNNRegressor` – an inference engine for Tensorflow neural networks performing regression,
- `TFNNClassifier` – an inference engine for Tensorflow neural networks performing *n*-ary classification, and
- `TFNNBinClassifier` – an inference engine for Tensorflow neural networks performing binary classification.

The evaluation of all ASCAPE machine learning models is enabled by the `evalm` module. This module defines three classes for computing various metrics reflecting the quality of a model for provided real (ground-truth) values:

- `EvalRegressionModel` for evaluating regression models. The following metrics are implemented in this class: mean absolute error (MAE), mean squared error (MSE), the coefficient of determination also known as the R^2 score and the Person correlation coefficient,
- `EvalClassificationModel` for evaluating classification models. This class provides methods returning values of the following metrics: accuracy, precision per class, recall per class, macro-averaged precision, macro-averaged recall, F1 score and confusion matrix, and
- `EvalBinClassificationModel` for evaluating binary classification models. This class is a simple specialization of the previous class for two fixed classes (positive and negative).

A Python demo program demonstrating the 10-fold cross validation of a neural network regression model on the Orebro dataset realized using previously described modules from the ASCAPE edge node machine learning core is shown in Figure 23.



```python
from loader import FoldedDataset
from tfnn import FedTFNNRegModel
from tfnnie import TFNNRegressor
from evalm import EvalRegressionModel

dataset = FoldedDataset("ORB-30-36.csv")
for fold in range(10):
    # load training data for the current fold
    predictors, target = dataset.getTrainingData(fold)

    # train the neural network model (10 layers, each containing 40 neurons)
    model = FedTFNNRegModel()
    model.setupTrainingData(predictors, target)
    model.createModelDefinition(
        numPredictors=predictors.shape[1],
        hiddenLayersStructure=[40] * 10,
        loss="mse", activation="relu"
    )
    model.update(numEpochs=200, batchSize=8)
    model.saveModel("tfregressor")

    # evaluate the neural network model on the test fold
    reg = TFNNRegressor("tfregressor")
    testPredictors, testTarget = dataset.getTestData(fold)
    e = EvalRegressionModel(reg, testPredictors, testTarget)
    print("Fold", fold, "MAE", e.getMAE())
```

```
svc@svcpc:~/ASCAPE/coreML$ python3 demo5.py
Fold 0 MAE 6.810308540816855
Fold 1 MAE 5.472507309495357
Fold 2 MAE 5.28537766975269
Fold 3 MAE 5.545176656622636
Fold 4 MAE 5.544833601566783
Fold 5 MAE 6.018228196261222
Fold 6 MAE 7.899576086747019
Fold 7 MAE 5.713395804689641
Fold 8 MAE 5.890836983396296
Fold 9 MAE 6.069320915019618
svc@svcpc:~/ASCAPE/coreML$ 
```

Figure 23. A Python program showing how to perform the 10-fold cross validation of a neural network model on the Orebro dataset using modules realizing the ASCAPE edge node core ML services.

The learning of a global model is done in the interaction between one or more federated learning clients running on ASCAPE edge nodes on the one side and the federated learning server (federated learning coordinator) running at the ASCAPE cloud at the other side. In the current implementation, federated learning clients and the federated learning server are exchanging JSON serialized Tensorflow neural network models using TCP/IP sockets after each epoch during the neural network training. Two federated learning modes are supported: (1) the incremental mode in which one federated learning client (ASCAPE edge node) solely creates and/or updates a global model, and (2) the semi-concurrent mode in which two or more federated learning clients at some interval in time concurrently perform model learning. The ASCAPE federated learning for a particular model always starts in the incremental mode and switches to the semi-concurrent mode when the federated learning server detects that there is more than one federated learning client working with that model.

The federated learning server performs multi-threaded and non-blocking communication with federated learning clients that is implemented in five Python modules:

- `ASCAPECloudFLManager` – this module defines the class of the same name containing methods to initialize the federated learning server, accept connections from federated learning clients and create appropriate communication and synchronization (coordination) threads.
- `SingleModelManager` – this module contains `SingleModelManager` class extending `Thread` class and realizing model synchronization threads. `ASCAPECloudFLManager` creates one model synchronization thread per each active model (a model becomes active with the first federated learning client working with it). One model synchronization thread coordinates all federated learning clients (their communication threads which are instances of `SingleConnectionThread` class described below) working with the corresponding model and performs the federated averaging process (averaging neural network weights) in the semi-concurrent federated learning mode.
- `SingleConnectionThread` – this module defines `SingleConnectionThread` class realizing client communication threads. `ASCAPECloudFLManager` creates one client communication thread for each active federated learning client. Client communication threads perform communication with federated learning clients (receiving and sending neural network weights).
- `util` – this module contains various helper functions to serialize and persist Tensorflow neural network models and functions for sending/receiving serialized models through Berkley sockets.
- `main` – this module contains the main function that creates and starts an instance of `ASCAPECloudFLManager` class.

The implementation of federated learning clients for Tensorflow neural network models is based on Tensorflow callbacks that are executed after each epoch during neural network training and after the model training ends. We have developed a generic module for building federated learning clients called `callbacks`. This module defines

`EdgeNodeMLCallbacks` class extending Tensorflow `Callback` class. Thus, instances of `EdgeNodeMLCallbacks` can be attached to any Tensorflow neural network model realized in the previously described `tfnn` module (`FedTFNNRegModel`, `FedTFNNClModel` and `FedTFNNBinClModel`). The constructor of `EdgeNodeMLCallbacks` makes a connection to the federated learning server (by creating a socket) and sends to it the identifier of the working model. The ASCAPE cloud federated learning manager checks whether the model corresponding to the received identifier exists among models persisted at the ASCAPE cloud. If the model exists then it sends model weights to the client and the model on the client is initialized with the received weights. Otherwise, the client sends the JSON-serialized model definition to the server. `EdgeNodeMLCallbacks` redefines two methods inherited from Tensorflow `Callback` class:

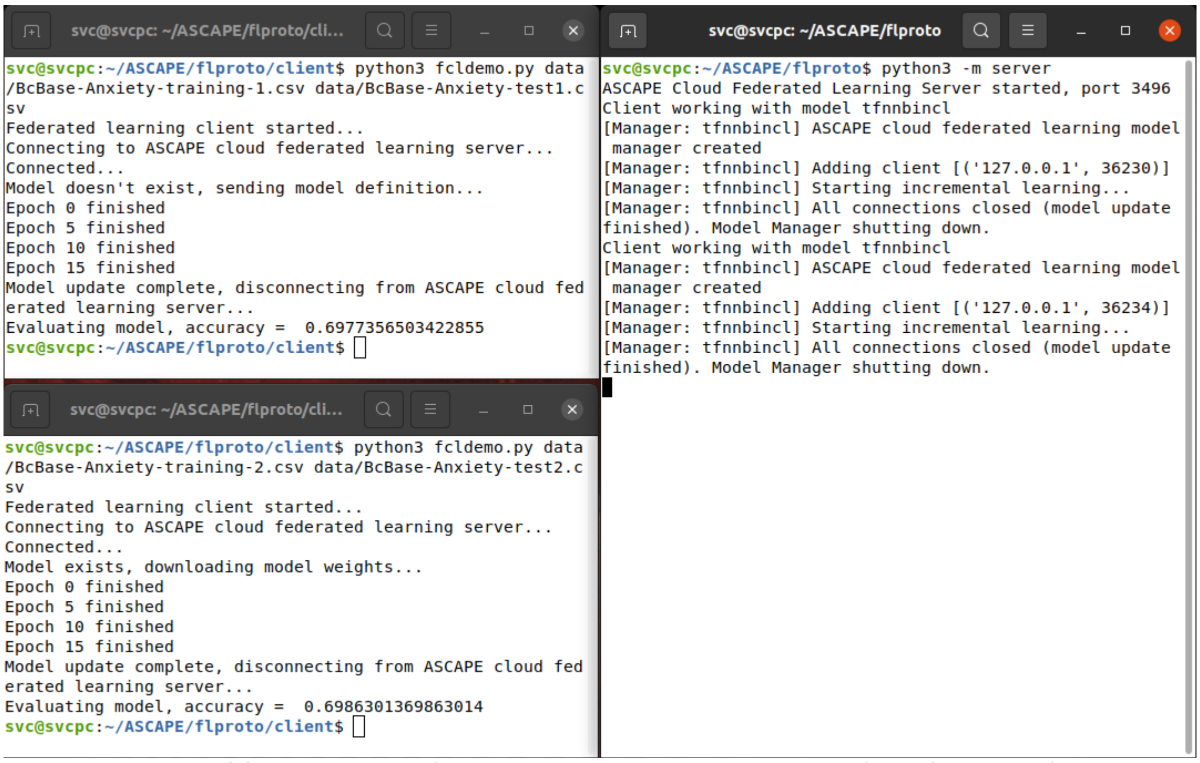
- `on_epoch_end` – this method, executed after each epoch during neural network training, serializes current neural network weights, sends them to the federated learning server and then waits for its response containing either a message to continue with current weights (the incremental learning mode) or new weights obtained after the federated averaging performed by the server (the semi-concurrent learning mode).
- `on_train_end` – this method, executed after the training ends, disconnects the clients from the server (by closing the socket created in the constructor).

A demo federated learning client realized using the previously described modules and classes is shown in Figure 24. The demo federated learning client has two arguments passed via the command line: the path to the training dataset on which a model called *tfnnbincl* is created or updated and the path to the test dataset on which the model is evaluated. The model is a Tensorflow neural network performing binary classification. The model has 5 hidden layers, each layer containing 10 neurons. The model is trained by ASCAPE edge nodes running in 20 epochs. The binary cross entropy is used as the loss function and model updates are performed after processing batches containing 16 instances from the training dataset. After creating an instance of the model, specifying the training dataset and specifying the structure of the model, a callback object is created and passed together with model learning hyperparameters to the update method called on the model. As already emphasized, the callback object handles communication with the federated learning server (in this example we assume that the federated learning server and clients execute on the same machine). When the federated learning process is finished, the model is saved and then loaded and evaluated using the appropriate inference engine (`TFNNBinClassifier`) and the appropriate model evaluation class (`EvalBinClassificationModel`).

```python
from loader import CSVDataset
from tfnn import FedTFNNBinClModel
from callbacks import EdgeNodeMLCallbacks
from tfnnie import TFNNBinClassifier
from evalm import EvalBinClassificationModel
import sys

if __name__ == '__main__':
    print("Federated learning client started...")
    trainingDataset = sys.argv[1]
    testDataset = sys.argv[2]
    tr = CSVDataset(trainingDataset)
    predictors, target = tr.getData()

    cl = FedTFNNBinClModel()
    cl.setupTrainingData(predictors, target)
    cl.createModelDefinition(
        numPredictors=predictors.shape[1],
        hiddenLayersStructure=[10] * 5, loss="binary_crossentropy")
    callback = EdgeNodeMLCallbacks(cl.model, "tfnnbincl",\
        host="127.0.0.1", port=3496)
    cl.update(numEpochs=20, batchSize=16, callback=callback)
    cl.saveModel("tfnnbincl")

    cl = TFNNBinClassifier("tfnnbincl")
    test = CSVDataset(testDataset)
    predictorsTest, targetTest = test.getData()

    ev = EvalBinClassificationModel(cl, predictorsTest, targetTest)
    print("Evaluating model, accuracy = ", ev.getAccuracy())
```

Figure 24. A demo federated learning client realized using the ASCAPE core ML services and the callbacks module.

To illustrate execution of the previously described demo federated learning client, we divided the BcBase dataset into four parts, two training datasets and two test datasets, for two federated learning clients. Then, we have executed federated learning clients (running on the same machine as the federated learning server, but as three independent processes, where clients communicate with the server using TCP/IP sockets) in two scenarios. In the first scenario (Figure 25) federated learning clients were executed sequentially (the second client was started after the first client finished), which corresponds to the incremental federated learning in which the first client created the model and the second client updated the model.


svc@svcpc: ~/ASCAPE/flproto/cli...
svc@svcpc: ~/ASCAPE/flproto

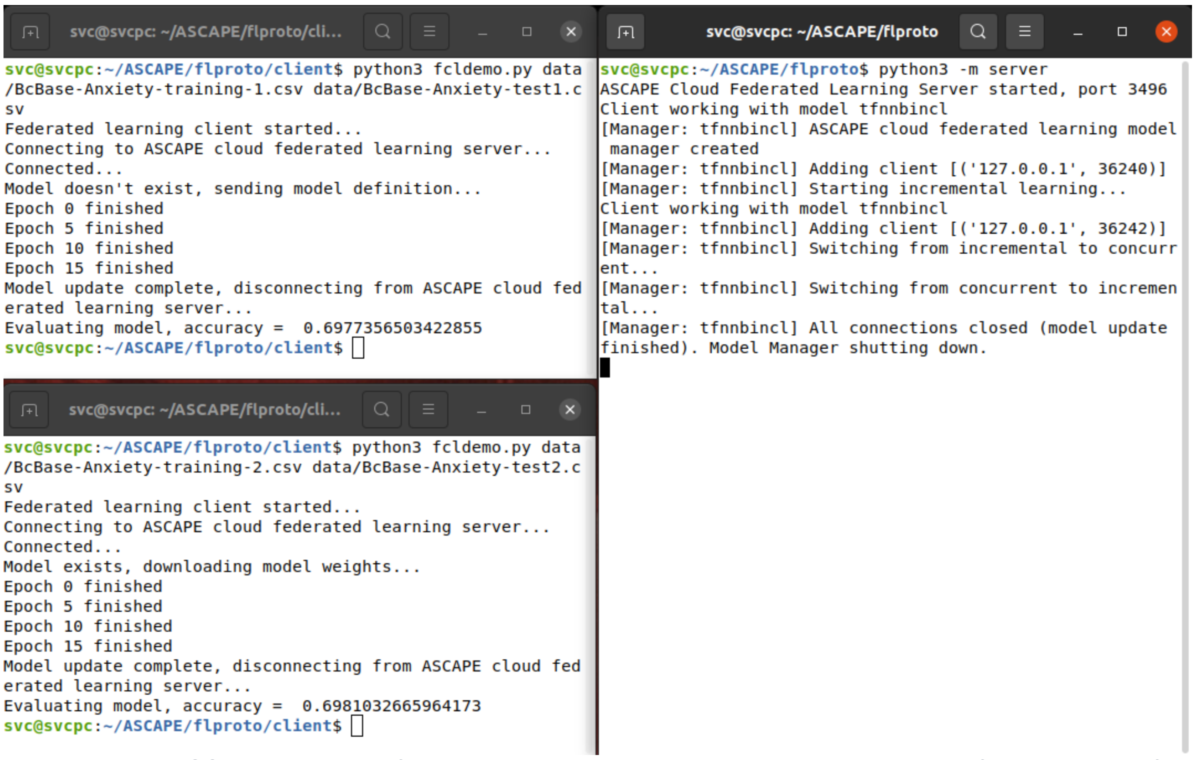
```
svc@svcpc:~/ASCAPE/flproto/client$ python3 fcldemo.py data
/BcBase-Anxiety-training-1.csv data/BcBase-Anxiety-test1.c
sv
Federated learning client started...
Connecting to ASCAPE cloud federated learning server...
Connected...
Model doesn't exist, sending model definition...
Epoch 0 finished
Epoch 5 finished
Epoch 10 finished
Epoch 15 finished
Model update complete, disconnecting from ASCAPE cloud fed
erated learning server...
Evaluating model, accuracy =  0.6977356503422855
svc@svcpc:~/ASCAPE/flproto/client$
```

```
svc@svcpc:~/ASCAPE/flproto/client$ python3 fcldemo.py data
/BcBase-Anxiety-training-2.csv data/BcBase-Anxiety-test2.c
sv
Federated learning client started...
Connecting to ASCAPE cloud federated learning server...
Connected...
Model exists, downloading model weights...
Epoch 0 finished
Epoch 5 finished
Epoch 10 finished
Epoch 15 finished
Model update complete, disconnecting from ASCAPE cloud fed
erated learning server...
Evaluating model, accuracy =  0.6986301369863014
svc@svcpc:~/ASCAPE/flproto/client$
```

```
svc@svcpc:~/ASCAPE/flproto$ python3 -m server
ASCAPE Cloud Federated Learning Server started, port 3496
Client working with model tfnnbincl
[Manager: tfnnbincl] ASCAPE cloud federated learning model
 manager created
[Manager: tfnnbincl] Adding client [('127.0.0.1', 36230)]
[Manager: tfnnbincl] Starting incremental learning...
[Manager: tfnnbincl] All connections closed (model update
finished). Model Manager shutting down.
Client working with model tfnnbincl
[Manager: tfnnbincl] ASCAPE cloud federated learning model
 manager created
[Manager: tfnnbincl] Adding client [('127.0.0.1', 36234)]
[Manager: tfnnbincl] Starting incremental learning...
[Manager: tfnnbincl] All connections closed (model update
finished). Model Manager shutting down.
```

Figure 25. Two ASCAPE edge node federating learning clients incrementally (one after another) training a federated tensorflow neural network on the BcBase dataset. Terminals running federated learning clients are shown on the left side, while the terminal running the ASCAPE cloud federated learning server is shown on the right side. Terminals running federated learning clients are shown on the left side, while the terminal running the ASCAPE cloud federated learning server is shown on the right side.

Figure 26 shows the second scenario corresponding to the semi-concurrent federated learning: the second client was started while the first client was running (after its first 5 epochs). It can be seen that in this scenario the federated learning server switched from the incremental learning mode to the semi-concurrent mode when the second client connected and returned back to the incremental mode when the first client finished.


svc@svcpc: ~/ASCAPE/flproto/cli...
svc@svcpc: ~/ASCAPE/flproto

```
svc@svcpc:~/ASCAPE/flproto/client$ python3 fcldemo.py data
/BcBase-Anxiety-training-1.csv data/BcBase-Anxiety-test1.c
sv
Federated learning client started...
Connecting to ASCAPE cloud federated learning server...
Connected...
Model doesn't exist, sending model definition...
Epoch 0 finished
Epoch 5 finished
Epoch 10 finished
Epoch 15 finished
Model update complete, disconnecting from ASCAPE cloud fed
erated learning server...
Evaluating model, accuracy =  0.6977356503422855
svc@svcpc:~/ASCAPE/flproto/client$
```

```
svc@svcpc:~/ASCAPE/flproto/client$ python3 fcldemo.py data
/BcBase-Anxiety-training-2.csv data/BcBase-Anxiety-test2.c
sv
Federated learning client started...
Connecting to ASCAPE cloud federated learning server...
Connected...
Model exists, downloading model weights...
Epoch 0 finished
Epoch 5 finished
Epoch 10 finished
Epoch 15 finished
Model update complete, disconnecting from ASCAPE cloud fed
erated learning server...
Evaluating model, accuracy =  0.6981032665964173
svc@svcpc:~/ASCAPE/flproto/client$
```

```
svc@svcpc:~/ASCAPE/flproto$ python3 -m server
ASCAPE Cloud Federated Learning Server started, port 3496
Client working with model tfnnbincl
[Manager: tfnnbincl] ASCAPE cloud federated learning model
 manager created
[Manager: tfnnbincl] Adding client [('127.0.0.1', 36240)]
[Manager: tfnnbincl] Starting incremental learning...
Client working with model tfnnbincl
[Manager: tfnnbincl] Adding client [('127.0.0.1', 36242)]
[Manager: tfnnbincl] Switching from incremental to concurr
ent...
[Manager: tfnnbincl] Switching from concurrent to incremen
tal...
[Manager: tfnnbincl] All connections closed (model update
finished). Model Manager shutting down.
```

Figure 26. Two ASCAPE edge node federating learning clients semi-concurrently training a federated tensorflow neural network on the BcBase dataset. Terminals running federated learning clients are shown on the left side, while the terminal running the ASCAPE cloud federated learning server is shown on the right side. Terminals running federated learning clients are shown on the left side, while the terminal running the ASCAPE cloud federated learning server is shown on the right side.

4.3 Homomorphic encryption: training and prediction

Homomorphic encryption is used as a privacy mechanism that enables both training and prediction on ASCAPE encrypted patient data. In this Section we present the current implementation of core ASCAPE machine learning services that enable the training of a global model on a collection of encrypted patient data, the prediction using the trained model and the evaluation.

The core ASCAPE HE-based machine learning services are enabled through an in-house developed C++ library that offers the necessary functions to train and test neural network-based models on ciphertext (MORE encrypted data obtained following the encryption algorithm presented in Figure 44). Moreover, a series of Python modules are implemented and offered for performance analysis. The HE setup, including ciphertext preparation, is presented in detail in Section 6.3.

The ASCAPE HE-based machine learning core contains the following main modules:

- `training` – a module for training and storing HE encrypted neural network-based machine learning models,
- `prediction` – a module realizing an inference engine (i.e., engine for making predictions on HE data) based on previously learned model.

The `training` module calls the `Training` function which is responsible for training a neural network-based machine learning model. The training module enables a MLP (multi-layer perceptron), whose configuration and hyperparameters are specified into a configuration file (config). The training algorithm is shown in Figure 27. The arguments needed to call this module are following:

```
    arg1: Training input (ciphertext) data file (path +.csv
filename)
    arg2: Training labels (ciphertext) data file (path +.csv
filename)
    arg3: Destination directory to save the model
    arg4: Config file (.json file)
```

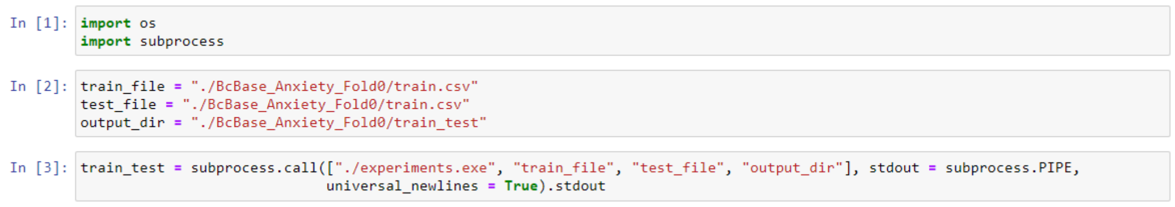
The `prediction` module calls the `Prediction` function which is responsible for performing predictions based on the neural network model previously trained. The prediction module required the configuration (config) file used by the training model to instantiate the model but also the weights learned during training. The prediction algorithm is shown in Figure 27. The arguments needed to call this module are following:

```
    arg1: Source input (ciphertext) data file (path +.csv
filename)
    arg2: Destination data file to store the encrypted
predictions (path +.csv
    arg3: Path to the saved the model
    arg4: Config file (.json file)
```

```
(1)  function TrainOnCiphertext( )
(2)      $\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}} \longleftarrow \text{LoadDataset}()$
(3)      $\mathbf{X}_{\text{train}} \longleftarrow \text{Normalize}(\mathbf{X}_{\text{train}})$
(4)      $\mathbf{S} \longleftarrow \text{KeyGeneration}()$
(5)      $\mathbf{X}_{\text{train}_{\text{enc}}} \longleftarrow \text{Encryption}(\mathbf{X}_{\text{train}}, \mathbf{S})$
(6)      $\mathbf{Y}_{\text{train}_{\text{enc}}} \longleftarrow \text{Encryption}(\mathbf{Y}_{\text{train}}, \mathbf{S})$
(7)      BuildModel()
(8)      $\text{Train}(\mathbf{X}_{\text{train}_{\text{enc}}}, \mathbf{Y}_{\text{train}_{\text{enc}}})$
(9)      return $\text{model}_{\text{enc}}$
(10) end function
(11) function PredictOnCiphertext( )
(12)     $\mathbf{X}_{\text{test}} \longleftarrow \text{LoadSamples}()$
(13)     $\mathbf{X}_{\text{test}} \longleftarrow \text{Normalize}(\mathbf{X}_{\text{test}})$
(14)     $\mathbf{S} \longleftarrow \text{LoadKey}()$
(15)     $\mathbf{X}_{\text{test}_{\text{enc}}} \longleftarrow \text{Encryption}(\mathbf{X}_{\text{test}}, \mathbf{S})$
(16)     LoadModel()
(17)     $\tilde{Y}_{\text{test}_{\text{enc}}} \longleftarrow \text{Predict}(X_{\text{test}_{\text{enc}}})$
(18)     $\tilde{Y}_{\text{test}} \longleftarrow \text{Decryption}(\tilde{Y}_{\text{test}_{\text{enc}}}, \mathbf{S})$
(19)     return $\tilde{Y}_{\text{test}}$
(20) end function
```

Figure 27. Neural network-based analysis on ciphertext data.

An example on how to call the C++ modules from Python is shown in Figure 28. Experiments.cpp contains the module calls training and prediction, as previously described.



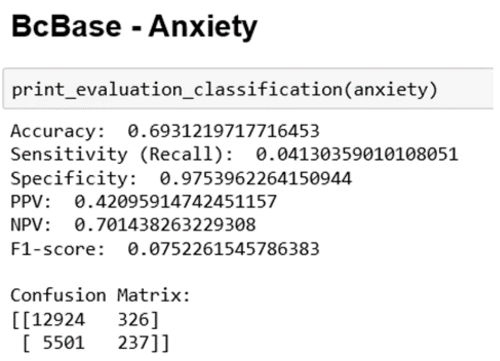
```
In [1]: import os
        import subprocess

In [2]: train_file = "./BcBase_Anxiety_Fold0/train.csv"
        test_file = "./BcBase_Anxiety_Fold0/train.csv"
        output_dir = "./BcBase_Anxiety_Fold0/train_test"

In [3]: train_test = subprocess.call(["./experiments.exe", "train_file", "test_file", "output_dir"], stdout = subprocess.PIPE,
                                     universal_newlines = True).stdout
```

Figure 28. Calling C++ modules.

Training of HE AI-based models has been performed on for two retrospective datasets: Breast cancer *BcBase* dataset and Prostate cancer *Orebro* dataset. The HE training setup and the performance evaluation are described in details in Deliverable 2.4. The performance of the encrypted models is performed after the decryption of the results. Evaluation metrics for these two datasets are implemented in the Jupyter Notebook *Metrics.ipynb*. This notebook also contains the functions for aggregating the folds' output. The list of this notebook's functions is listed below:

- *denormalize* – denormalizes the predictions using the formula:
$$y_{denormalized} = y_{pred}\,(\max(x) - \min(x)) + \min(x)$$
- *regression_predictions* – concatenates the predictions for the Orebro datasets
- *regression_stratistics* – calculates the regression statistics
- *BA_plot* – plots the Bland Altman plot
- *Scatter_plot* – plots the Scatter plot
- *total_confusion_matrix* – returns the sum of the confusion matrixes of each fold of the BcBace datasets
- *classification_metrics* – calculates the binary classification metrics
- *print_evaluation_regression* – prints statistics and plots for the regression problem (Orebro)
- *print_evaluation_classification* – prints metrics and confusion matrix for the binary classification problem (BcBase)

The *Results.ipynb* notebook calls the corresponding functions and prints the final results for each 10 main datasets. Figure 29 shows an example for the BcBase Anxiety dataset.



BcBase - Anxiety

```
print_evaluation_classification(anxiety)

Accuracy:  0.6931219717716453
Sensitivity (Recall):  0.04130359010108051
Specificity:  0.9753962264150944
PPV:  0.42095914742451157
NPV:  0.701438263229308
F1-score:  0.0752261545786383

Confusion Matrix:
[[12924   326]
 [ 5501   237]]
```

Figure 29. Results.jpynb example – BcBase Anxiety.

4.4 Implementation technologies and PoC repositories

The ASCAPE core machine learning services, enabling training and instrumentation of predictive global and local models by ASCAPE Edge AI Models Manager and ASCAPE Edge AI Predictions & Simulations Manager, are developed in the programming language Python using three free and open-source machine learning libraries:

1. Scikit-learn [9] – a library providing various classification, regression and clustering algorithms designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. The library additionally provides classes and functions used to implement the validation procedures for ASCAPE predictive models.
2. Tensorflow [12] – a machine learning and symbolic math library developed by the Googe Brain Team that is based on dataflow and automatic differentiation principles enabling training and inference of deep neural networks.
3. Keras [10] – a library focused on artificial neural networks. Keras acts as an interface for the abovementioned Tensorflow library.

The source code of ASCAPE core machine learning services is available at the following ASCAPE GitLab repository: https://gitlab.com/ascape-h2020/machine-learning/coreedgeml. The source code of the federated learning server (part of Cloud Federated Learning Coordinator) and the federated learning client (part of Edge AI Models Manager) prototypes can be found at: https://gitlab.com/ascape-h2020/machine-learning/fl-prototype.

The ASCAPE core machine learning services on HE data makes use of CipherML – an in-house developed library for privacy-preserving machine learning, which is based on the MORE encryption scheme. The library can be used to train standard neural network models, but also to make inferences (predictions) based on MORE homomorphically encrypted data. The library offers a high-level Keras-like API that enables fast prototyping privacy-preserving neural network models with minimal code. Written in the C++ programming language, the CipherML allows learning-based models to be defined by stacking one building block on top of the other, and then trained on MORE homomorphically encrypted data by simply calling the fit method. Similarly, prediction can be performed using the predict method.

The evaluation of the trained models was performed in Python 3.8 using SciKit-Learn package, Matplotlib-package. The source code for model evolution is available at the following ASCAPE GitLab repository: https://gitlab.com/ascape-h2020/machine-learning/a3-model-evaluation/-/tree/master/Homomorphic%20Encryption.

5 Cancer care predictive Explainable AI

All QoL predictions made by any model in the ASCAPE platform are critical information that could influence the decision of which medical interventions will be done. Therefore, every prediction must be provided with information on how the decision was made, with what data it was made and how accurate the prediction is.

5.1 Explainable AI

The accuracy (and therefore reliability) of a model is determined during model evaluation after the training process. For regression models, the mean absolute error is a good performance indicator. If the predictions of a model are following a normal distribution, the MAE can be interpreted like the standard deviation. The calculated error can then be provided for every prediction the model is used for. When retraining the model, it is evaluated again, and the MAE is updated as well. For classification tasks with neural networks, the weights of the output vector can be used as an indicator for the reliability. All weights of the output vector sum up to 1 if a softmax-activation is used. To make sure a weight corresponds to the probability that the classification is correct, the softmax-layer's temperature must be calibrated, although this is trivial to do. For all classification models that are not neural networks, Scikit-learn provides a function called *predict_proba* with the same functionality.

ASCAPE makes use of different machine learning techniques. Some of them, like artificial neural networks, are based on complex algorithms processing high-dimensional data and cannot be easily described. Therefore, surrogate models are used. Surrogate models are machine learning models that are interpretable by design (e.g., decision trees or linear regressors). For each model used in ASCAPE, a dedicated surrogate model is trained to provide explanations for its predictions. They are trained on the same training input data that the regular model is trained on but instead of using the ground truth as labels, the output of the regular model is used. This way they mimic the predictions of the model used to make a prediction. Their decision process can then be used to explain a prediction or to provide a ruleset describing the inference of models' outputs that is too complex to be understood by a human. In ASCAPE, linear regression, logistic regression and decision trees are used as surrogate models. Decision trees can be visualized and used for a human readable rule extraction. The coefficients of a linear/logistic regressor describe linear dependencies between the input and the output of a model. A full description and evaluation of trained surrogate models can be found in Deliverable 2.4, section 5.3.

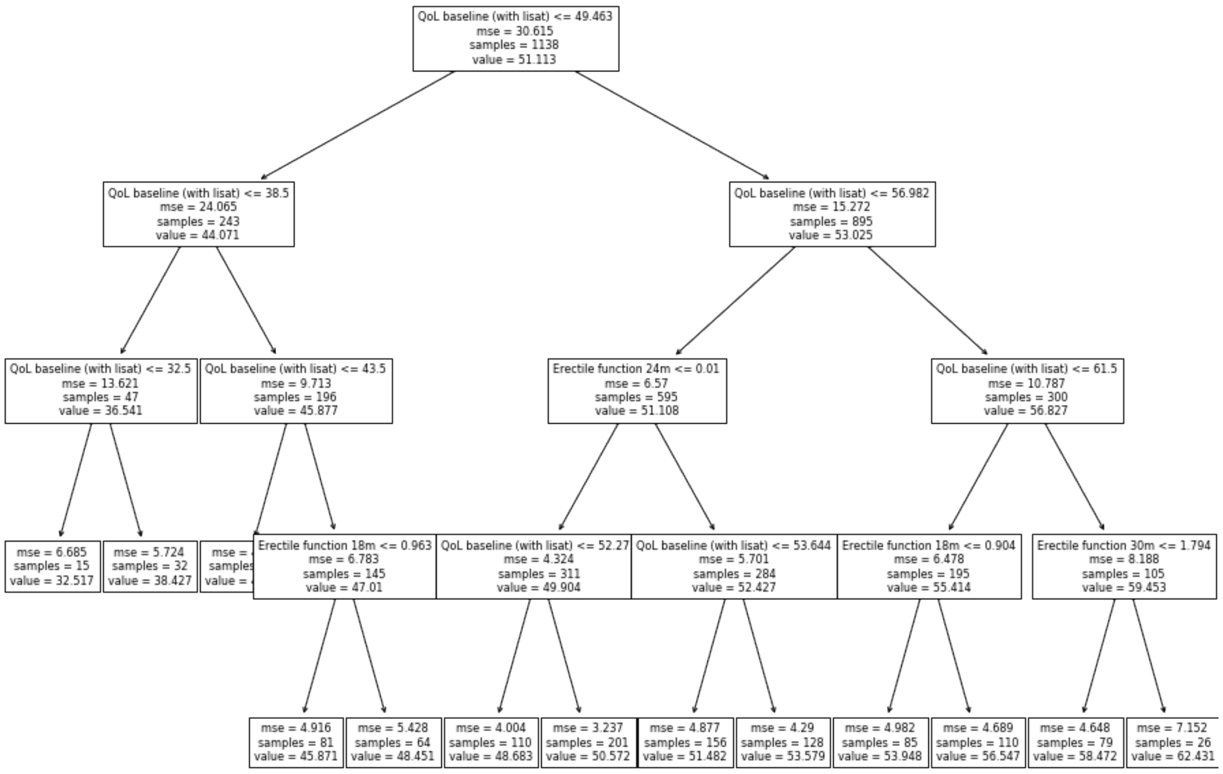


Figure 30. Visualization of a surrogate decision tree model. The top line in each box describes the split condition.

Feature attributions describe how strongly each input variable influences the models output. There are various concepts to calculate feature attribution like Permutation Importance, LIFT, Integrated Gradients and SHAP. In ASCAPE, we decided to use SHAP, because it is based on Shapley values, a game-theoretic approach, which is mathematically founded and can therefore be consistently calculated for different model types. It satisfies the properties of Additive Feature Attribution, which means that the average expected model output φ_0 plus the calculated Feature Attributions φ_i sum up to the model output *f(x)*:

$$f(x) = \phi_0 + \sum_{i=1} \phi_i$$

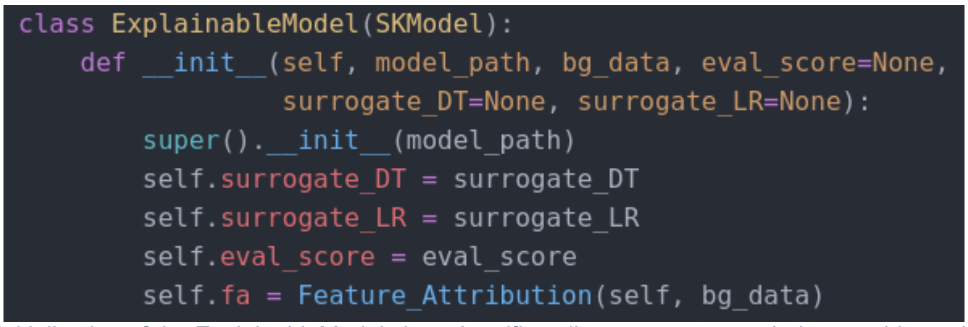
We will use the same-named python library SHAP [3], which also makes use of other state-of-the-art feature attribution techniques to accelerate the computation. Figure 32 shows the implementation of a FeatureAttribution class, which internally uses SHAP's *KernelExplainer* class to calculate model-agnostic Feature Attributions for predicitons. The SHAP-library also contains concepts and implementations to visualise feature attribution values in an intuitive way as shown in Figure 33. It shows a bar plot of the mean absolute feature attributions of a ridge regression model trained on the ORB-30-120 dataset.

We implemented an explainable model class, which is a wrapper class for the regular model inference class (Figure 31). It provides a set of methods to infer explanations with any underlying AI model. For each prediction ASCAPE can provide:

- Confidence levels of the prediction
- The decision path of the decision tree surrogate model
- Feature attributions for each input feature

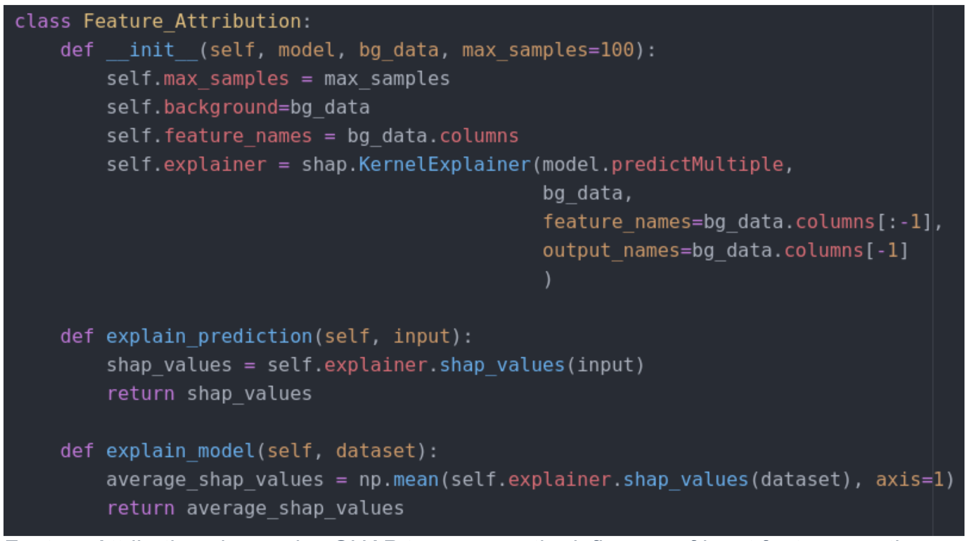
For any model, ASCAPE can provide:

- Overall Feature attribution of the training dataset (See Figure 33)
- Visualization of the surrogate decision tree (See Figure 30)
- Coefficients of the surrogate linear regressor



```python
class ExplainableModel(SKModel):
    def __init__(self, model_path, bg_data, eval_score=None,
                 surrogate_DT=None, surrogate_LR=None):
        super().__init__(model_path)
        self.surrogate_DT = surrogate_DT
        self.surrogate_LR = surrogate_LR
        self.eval_score = eval_score
        self.fa = Feature_Attribution(self, bg_data)
```

Figure 31. Initialisation of the ExplainableModel class. It unifies all components needed to provide explanations for every prediction



```python
class Feature_Attribution:
    def __init__(self, model, bg_data, max_samples=100):
        self.max_samples = max_samples
        self.background=bg_data
        self.feature_names = bg_data.columns
        self.explainer = shap.KernelExplainer(model.predictMultiple,
                                              bg_data,
                                              feature_names=bg_data.columns[:-1],
                                              output_names=bg_data.columns[-1]
                                              )

    def explain_prediction(self, input):
        shap_values = self.explainer.shap_values(input)
        return shap_values

    def explain_model(self, dataset):
        average_shap_values = np.mean(self.explainer.shap_values(dataset), axis=1)
        return average_shap_values
```

Figure 32. Feature Attribution class using SHAP to measure the influence of input features on the model's output

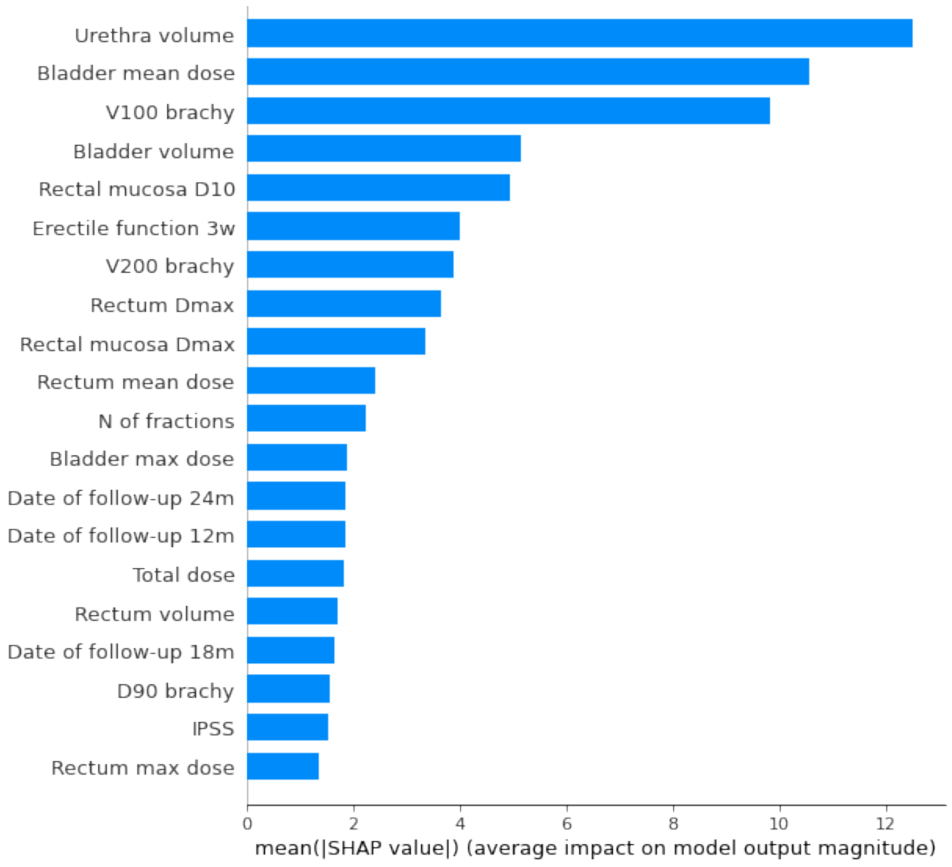


Figure 33. Bar plot showing the average absolute feature attribution of a test dataset.

5.2 Simulations

A central objective of ASCAPE is to actively propose medical interventions that potentially improve the patient's future quality of life. The machine learning models themselves can only infer estimations of the QoL based on a patient's medical data. To identify medical interventions, changes in the input data are made, passed through the model and the predictions are recorded.

Currently, the following steps are executed to perform simulations and identify promising treatments:

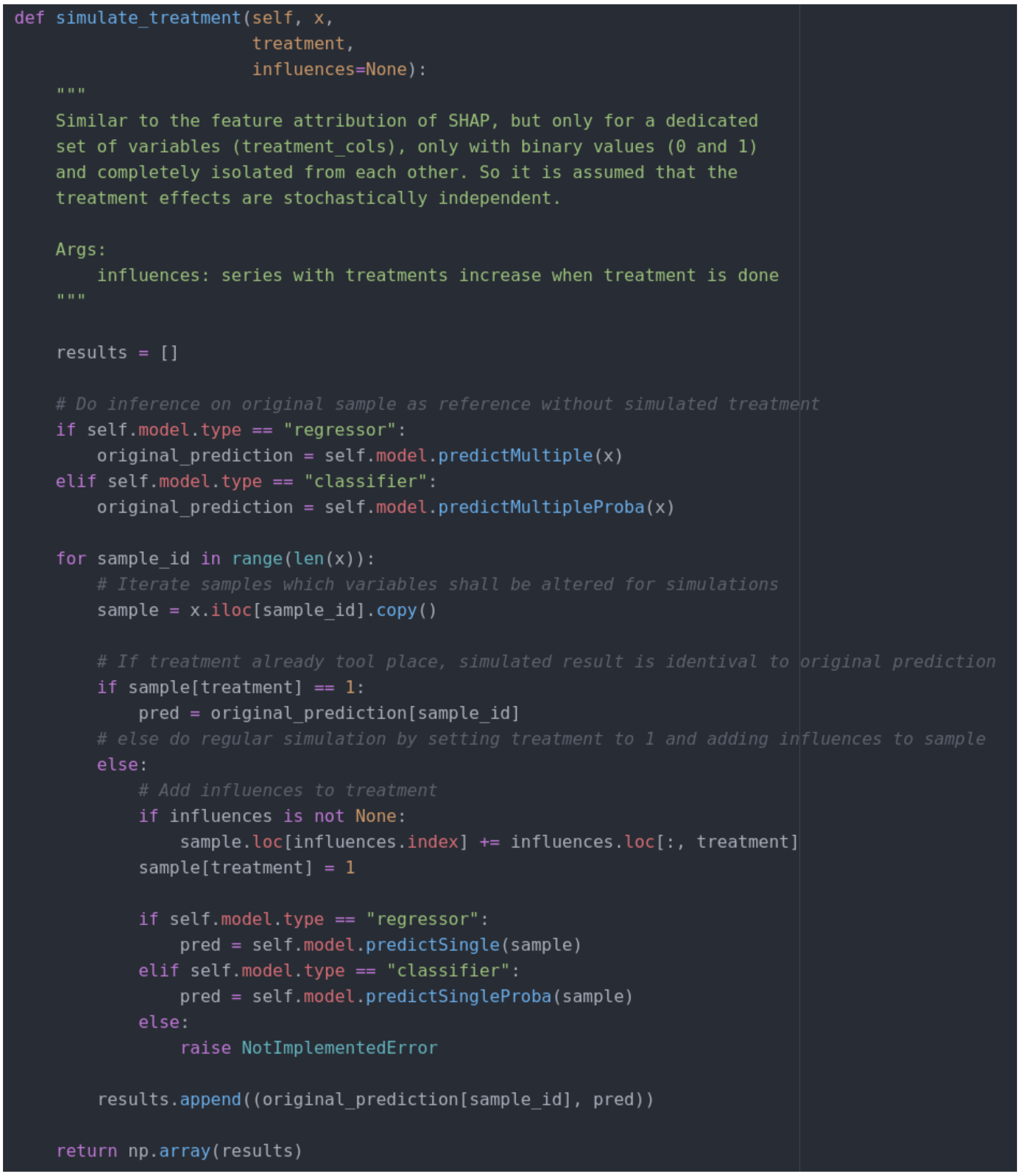
First, Feature Attribution is used to identify variables that strongly influence the QoL prediction. This way features that have little influence on the output can be filtered and are not changed during simulations.

The patient data contains proxy variables indicating if a certain treatment was done. If a treatment was done, it will influence other variables in the patient's medical data. These influences must be identified. As of now, only retrospective datasets are available, which only contain medical data that was collected once. The datasets are therefore lacking information on how the treatments influence the medical data. Therefore, the second step is the segmentation of the dataset into two cohorts for each treatment. One cohort contains all patients who did receive the respective treatment and the other did not. The influence of a treatment is then calculated as the difference between the variables' mean values of the two cohorts.

Once prospective datasets are available, this step will be re-evaluated and may be optimized to identify more accurate causalities from interventions to other medical variables. For retrospective datasets, accuracy might be further improved with matched cohorts [14].

Third, for each treatment proxy that equals 0, i.e., every treatment that was not done yet, all other variables are changed according to the influences calculated in the second step and the treatment proxy is set to 1. This way hypothetical patient data is created in which a medical intervention has been taking place. The data is passed into the AI model and the QoL prediction is recorded. This step gets executed for each treatment proxy individually. Figure 34 shows the implementation of this step. The function *simulate_treatment* expects a sample containing patient data that shall be simulated, the treatment to be simulated and an array containing the expected changes in health data once the treatment took place. A more thorough explanation and more examples can be found in deliverable D2.4, Section 6.3.

Last, the treatment that increased the predicted QoL the most is highlighted as a recommended intervention to the ASCAPE user. Regarding explainability, the expected increase in QoL, the accuracy of the model and the calculated feature attributions of the first step are shown as well.



```python
def simulate_treatment(self, x,
                       treatment,
                       influences=None):
    """
    Similar to the feature attribution of SHAP, but only for a dedicated
    set of variables (treatment_cols), only with binary values (0 and 1)
    and completely isolated from each other. So it is assumed that the
    treatment effects are stochastically independent.

    Args:
        influences: series with treatments increase when treatment is done
    """

    results = []

    # Do inference on original sample as reference without simulated treatment
    if self.model.type == "regressor":
        original_prediction = self.model.predictMultiple(x)
    elif self.model.type == "classifier":
        original_prediction = self.model.predictMultipleProba(x)

    for sample_id in range(len(x)):
        # Iterate samples which variables shall be altered for simulations
        sample = x.iloc[sample_id].copy()

        # If treatment already tool place, simulated result is identival to original prediction
        if sample[treatment] == 1:
            pred = original_prediction[sample_id]
        # else do regular simulation by setting treatment to 1 and adding influences to sample
        else:
            # Add influences to treatment
            if influences is not None:
                sample.loc[influences.index] += influences.loc[:, treatment]
            sample[treatment] = 1

            if self.model.type == "regressor":
                pred = self.model.predictSingle(sample)
            elif self.model.type == "classifier":
                pred = self.model.predictSingleProba(sample)
            else:
                raise NotImplementedError

        results.append((original_prediction[sample_id], pred))

    return np.array(results)
```

Figure 34. Simulation of patient data changes after a treatment and inference of predictions

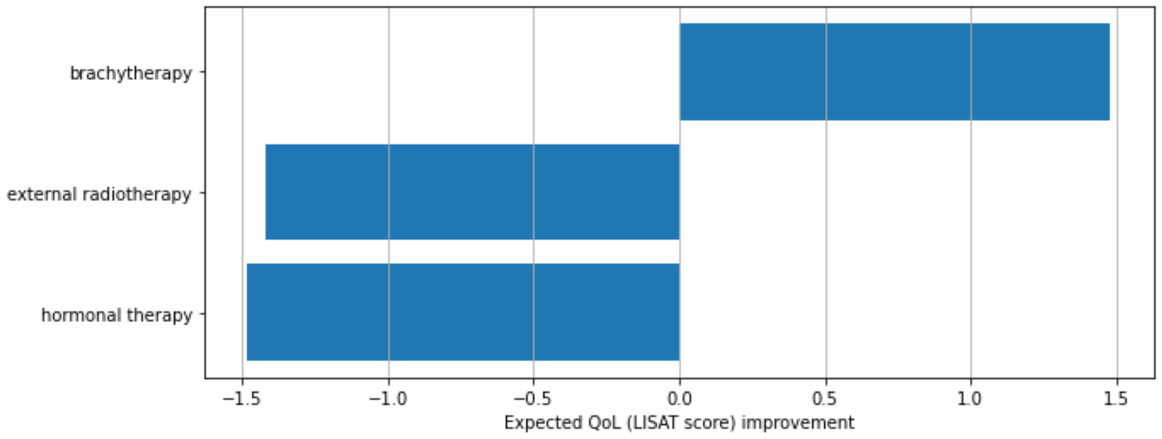


Figure 35. Example of a simulation result. Only brachytherapy is expected to improve the QoL of the patient.

5.3 Implementation technologies and PoC repositories

Both Explainability and Simulations were implemented using Python 3.8. For data management, the Pandas-package was used. Plots were generated using the Matplotlib-package. As described in Section 5.1, the SHAP-package was used to calculate Feature Attributions. The surrogate models were trained using SciKit-Learn [1].

The code for the Proof of Concept is available in two repositories in the ASCAPE GitLab repository. *A4-Explainability* contains the wrapper class for explainable AI, surrogate training and evaluation (Link: https://gitlab.com/ascape-h2020/machine-learning/a4-explainability). *A5-Simulations* contains all the code to run the simulations itself (Link: https://gitlab.com/ascape-h2020/machine-learning/a4-explainability). Since Feature Attribution techniques are used for it, this repository is referenced as a submodule in *A4-Explainability*.

6 ASCAPE Security and privacy toolkit

In the context of ASCAPE, mechanisms will be designed and developed to ensure the security and privacy of the data held in the platform and the integrity of the platform itself. A security and privacy toolkit that consists of a security management mechanism (Security Gatekeeper), an encryption block (Homomorphic Encryption) and a privacy block (Differential Privacy) will be deployed. The work carried out concerning those three mechanisms for the Proof of Concept is described below.

6.1 Security Management

The ASCAPE Gatekeeper is an ASCAPE component that enables authentication and acts as an API gateway. The tool, integrated in the ASCAPE framework to offer these services, is WSO2 [15] and the products, chosen for the ASCAPE framework and will be demonstrated as a proof of concept in M14, are:

- WSO2 Identity Server (IS) [16]
- WSO2 API Manager (APIM) [17]

Those two products were first evaluated, based on a detailed state of the art analysis of solutions available in the open-source ecosystem.

6.1.1 WSO2 Identity server

WSO2 IS provides secure identity and access management by managing identity of the users efficiently and by facilitating the centralized management, administration, and monitoring.

6.1.1.1 Architecture elements

WSO2 IS, despite being a commercial off-the-shelf packaged solution (not a custom-made solution), enables easy customisation and extension through its componentised architecture. The WSO2 IS can be used directly by administrators (or other users if proper authorisation is provided), through its Management Console. Apart from the registered users, IS can be used as an identity provider for third party systems that have their own set of users.

The Identity Server is using inbound authenticators. An inbound authenticator parses all the incoming authentication requests and, if they can be handled, converts them into a format understood by the IS authentication framework and passes them to the authentication framework. It then constructs corresponding responses for all the supported protocols and passes along the response to the calling party. SAML 2.0, OAuth 2.0 and Open Id Connect are some of the supported authentication protocols.

6.1.1.2 Customisation

In order to use the services provided by the IS, an ASCAPE component needs to be registered in it. The registration procedure of an Open Id Connect (OIDC) - OAuth 2.0 Relying Party (client) consists of the actions listed below:

1. Registration of the Relying Parties in the IS where the callback URL and Relying Party (client) name need to be provided.
2. Communication of Client ID and Client Secret to the Relying Party.

The registration procedure of a Service Provider using the SAML 2.0 authentication protocol follows the actions listed below:

1. Registration of the Service Provider in the IS.
2. Metadata exchange between IS and the Service Provider.

6.1.1.3 Authorisation Flow

According to OAuth2.0 specification [8] the Resource owner (API owner) is responsible for either granting access to authorized users or preventing unauthorized users from accessing a resource (an API). For that to happen:

1. The API consumers send a valid OAuth2.0 access token, as HTTPS request parameter, along with every API call.
2. Once the API Server receives the call, it validates the access token against the OAuth2.0 Authorization server.
3. If the token is valid, it processes the request and replies to the consumer API. Otherwise, the server returns an error response.

6.1.2 WSO2 API manager

The WSO2 APIM's main features are design and prototyping for SOAP or RESTful APIs, governance policies and access control with OAuth2. It accepts OpenAPI (formerly Swagger) specifications [18] and manages throttling, access level and integration with Identity provider.

6.1.2.1 Overview

WSO2 API Gateway provides a runtime and a backend component (an API proxy) for API calls. It secures, protects, manages, and scales API calls by intercepting API requests and applying policies, such as throttling and security, using handlers and managing API statistics. Upon validation of a policy, the Gateway passes Web service calls to the actual backend. If the service call is a token request, the Gateway passes it directly to the Key Manager. After the API Manager server has started, you can access the Gateway using the Management Console.

6.1.2.2 Message Flow

Messages that reach the Gateway are processed as follows:

1. When a request hits the API Gateway, it is received by the HTTP/HTTPS transport that is responsible for carrying messages in a specific format. The transport provides a receiver and a sender (for receiving and sending messages accordingly).
2. The receiving transport selects a message builder, based on the message's content type, and uses the selected one in order to process the message's raw

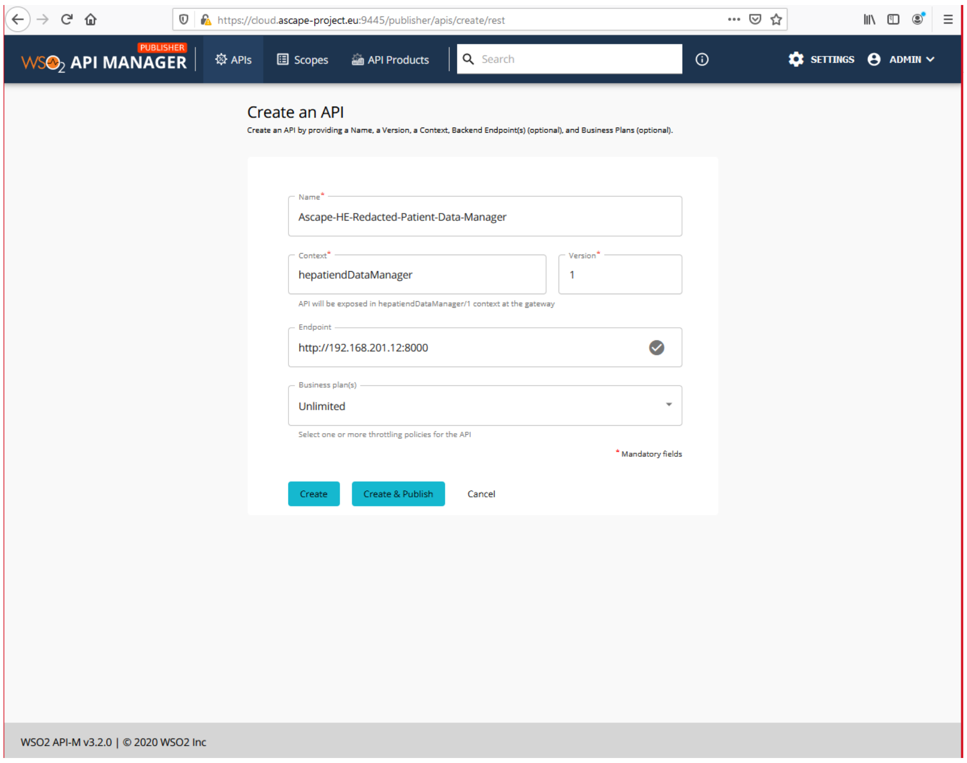
payload data and convert it into a common XML, which the Gateway mediation engine can then read and understand.

3. The request is passed through a set of handlers that applies the quality of services on the request message. Furthermore, it enforces security rate-limiting, and transformations on API requests if applicable.
4. After all the requests are routed to the backend endpoint, a message formatter (selected based on the message's content type) is used to build the outgoing stream back into its original format based on the message.
5. The transport sends the message out from the Gateway.

6.1.3 Cloud Server Installation

The WSO2 APIM was deployed on a VM in Openstack running on the Cloud Server. For the purpose of the Proof of Concept demonstration, docker and docker compose were installed. In the ASCAPE Gitlab a docker-compose.yml file is provided that, upon its execution, creates all the containers needed to run the demo.

The url *https://cloud.ascape-project.eu* will redirect to the login page and, once the credentials are inserted, the WSO2 API Publisher will enable the design of a new REST API (Figure 36) or the use of an existing REST API. After adding the apicdoc url and the endpoint, the API should get published in order to be visible in the Developer Portal and available for subscription. After the keys are generated, the API can be invoked using curl. The generation of keys is presented in Figure 37 and the curl command that runs, in order to execute the REST API call is presented in Figure 38. Since this is a PoC demonstrator, there is no SSL certificate issued yet for the website that hosts the demonstrator, so the web browser has the not-secure indication visited.


https://cloud.ascape-project.eu:9445/publisher/apis/create/rest

Create an API

Create an API by providing a Name, a Version, a Context, Backend Endpoint(s) (optional), and Business Plans (optional).

Name*: Ascape-HE-Redacted-Patient-Data-Manager

Context*: hepatiendDataManager

Version*: 1

API will be exposed in hepatiendDataManager/1 context at the gateway

Endpoint: http://192.168.201.12:8000

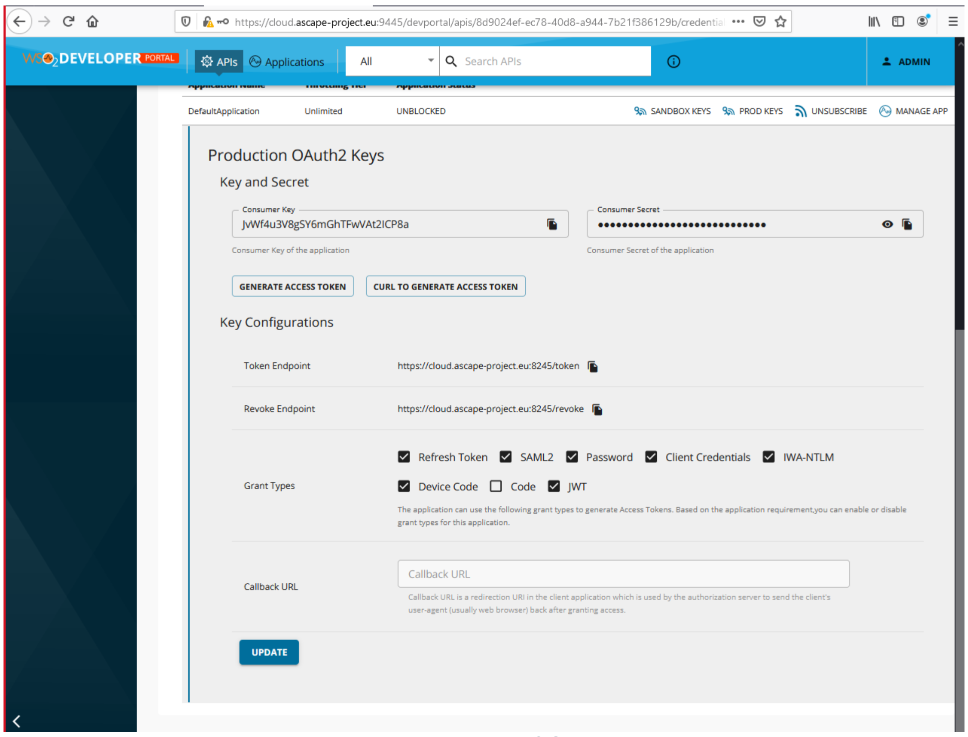
Business plan(s): Unlimited

Select one or more throttling policies for the API

* Mandatory fields

WSO2 API-M v3.2.0 | © 2020 WSO2 Inc

Figure 36. REST API creation


https://cloud.ascape-project.eu:9445/devportal/apis/8d9024ef-ec78-40d8-a944-7b21f386129b/credentia

Application Name	Throttling Tier	Application Status
DefaultApplication	Unlimited	UNBLOCKED

Production OAuth2 Keys

Key and Secret

Consumer Key: JvWf4u3V8gSY6mGhTFwVAt2ICP8a

Consumer Key of the application

Consumer Secret: ••••••••••••••••••••••••••••

Consumer Secret of the application

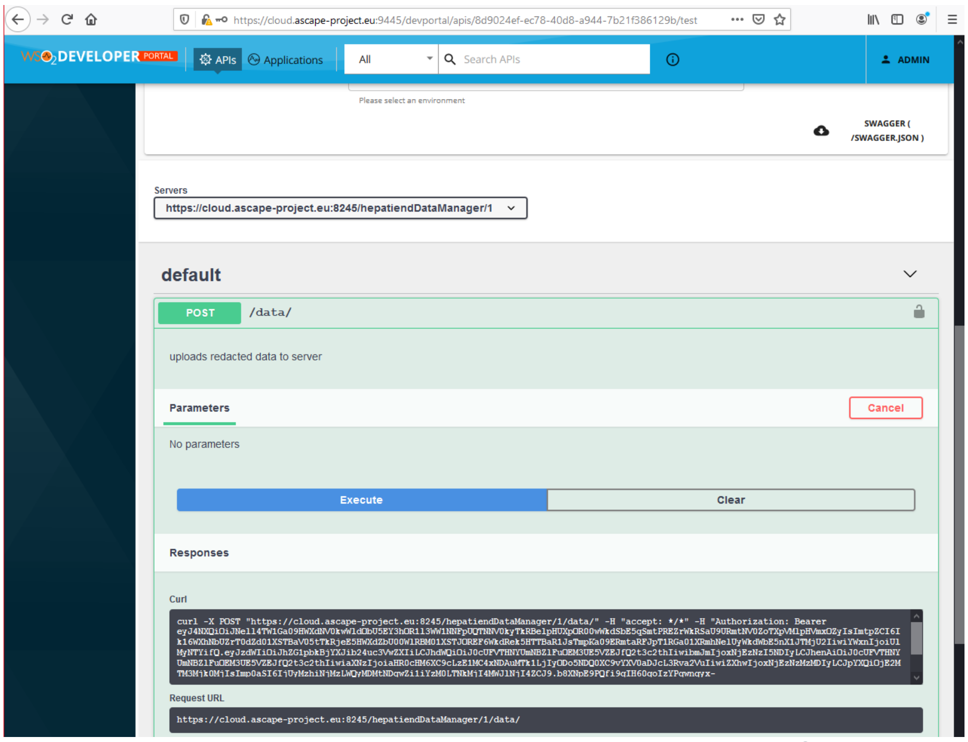
Key Configurations

Token Endpoint	https://cloud.ascape-project.eu:8245/token
Revoke Endpoint	https://cloud.ascape-project.eu:8245/revoke
Grant Types	✓ Refresh Token ✓ SAML2 ✓ Password ✓ Client Credentials ✓ IWA-NTLM ✓ Device Code ☐ Code ✓ JWT
Callback URL	Callback URL

The application can use the following grant types to generate Access Tokens. Based on the application requirement,you can enable or disable grant types for this application.

Callback URL is a redirection URI in the client application which is used by the authorization server to send the client's user-agent (usually web browser) back after granting access.

Figure 37. Production of OAuth2 Keys



uploads redacted data to server

No parameters

```
curl -X POST "https://cloud.ascape-project.eu:8245/hepatiendDataManager/1/data/" -H "accept: */*" -H "Authorization: Bearer
eyJ4NXQiOiJNell4TW1Ga09HWXdNV0kwWldObU5EY3hOR113WW1NNFpUQTNNV0kyTkRBelpHUXpOR00wWkdSbE5qSmtPREZrWkRSaU9URmtNV0ZoTXpVMlpHVmxOZyIsImtpZCI6I
k16WXhNbUZrT0dZd01XSTBaV05tTkRjeE5HWXdZbU00WlRBM01XSTJOREF6WkdReE5HTTBaR1JsTmpKa09ERmtaRFJpT1RGa01XRmhNelUyWkdWbE5nX1JTMjU2IiwiYWxnIjoiUl
MyNTYifQ.eyJzdWIiOiJhZG1pbkBjYXJib24uc3VwZXIiLCJhdWQiOiJ0cUFVTHNYUmNBZ1FuOEM3UE5VZEJfQ2t3c2thIiwibmJmIjoxNjEzNzI5NDIyLCJhenAiOiJ0cUFVTHNY
UmNBZ1FuOEM3UE5VZEJfQ2t3c2thIiwiaXNzIjoiaHR0cHM6XC9cLzE1MC4xNDAuMTk1LjIyODo5NDQ0XC9vYXV0aDJcL3Rva2VuIiwiZXhwIjoxNjEzNzMzMDIyLCJpYXQiOjE2M
TM3Mjk0MjIsImp0aSI6IjUyMzhiNjMzLWQyMDMtNDgwZi1iYzM0LTNkMjI4MWJlNjI4ZCJ9.b8XNpE9PQfi9qIH60qoIzYPqwnqyx-
```

```
https://cloud.ascape-project.eu:8245/hepatiendDataManager/1/data/
```

Figure 38. The curl command that runs in order to execute the REST API

6.2 Differential privacy

Within the ASCAPE project, DP will be implemented by adding a controlled amount of noise (the noise which follows Laplacian distribution) to the training data before the training process. In such a way, the whole learning model will preserve the desired amount of privacy. The more detailed explanation of DP concept is explained in the deliverable D2.4 which is submitted in parallel with this one.

This demonstration will introduce a privacy preserving parameter epsilon and illustrate how this parameter influences privacy. Also, this parameter influences the accuracy of the models, and this demonstration is also presented here. The extensive evaluation on all retrospective datasets and with several prediction models is not feasible in this timeframe, so the proof-of-concept demonstration will focus on one dataset and on several prediction models. This section will present the key features of the implemented DP component, while deliverable D2.4 will contain the detailed evaluation and results of this component.

The component for handling DP within ASCAPE project is `DP_dataset.py`. This component contains two functions which are responsible for the addition of noise to the input dataset. This dataset should not contain missing values, which means that it accepts datasets previously treated by the missing value inference module described in Section 4.1. The output database is in the same format as the input database, but with added noise to the majority of features.

The function which essentially calculates the noise is `laplaceMechanism()` which is shown in Figure 39. It has two parameters: (1) the value on which the noise should be added and (2) the epsilon parameter. The output of the function is the input value with added noise which draws samples from the Laplace distribution whose scale (decay) is *1/ε*.

```
import os
import pandas as pd
import numpy as np

def laplaceMechanism(x, epsilon):
    x +=  np.random.laplace(0, 1.0/epsilon, 1)[0]
    return x
```

Figure 39. Function `laplaceMechanism`.

The function `add_noise()` performs noise addition on the whole dataset (Figure 40.). The first two parameters are the names of input and output files. The third parameter represents the set of features (columns) on which the noise addition is not applied. Those features should include: categorical features, Boolean features, one-hot-encoding features, class features, etc. The last parameter represents the value of DP epsilon parameter. The function performs the noise addition on the values of all features which are not listed in `exclude_set` from the input file. The result is saved in the output file.

```
def add_noise(in_file, out_file, exclude_set, epsilon):
    if not os.path.exists(in_file):
        print('Error: Path "' + in_file + '" does not exist.')
        return

    dataset = pd.read_csv(in_file, ",")

    for i in dataset.axes[0]:
        for j in dataset.axes[1]:
            if j not in exclude_set:
                dataset.at[i, j] = laplaceMechanism(dataset.at[i, j], epsilon)

    dataset.to_csv(out_file, index=False)
```

Figure 40. Function `add_noise`.

Figure 41. shows the fragment of code which applies 10 values of ε-noise on the Orebro dataset. Note that `exclude_set` contains 21 feature names for this dataset, and that not all values are shown in figure.

```
exclude_set = {"T stage_1", "T stage_2", "T stage_3", "T stage_4", "N stage_0", "N stage_1", "N
epsilons = [0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100]
for epsilon in epsilons:
    add_noise("ORB-30-36.csv", "ORB-30-36_DP_eps_"+str(epsilon)+".csv", exclude_set, epsilon)
```

Figure 41. A Python program illustrating how to perform noise addition on dataset.

As already mentioned, the output database is in the same format as the original input database which is prepared with the missing inference component (described in Section 4.1). Accordingly, the same evaluation methodology as described in Section 4.2 could be applied here.

The main challenge in the introduction of privacy preserving techniques in machine learning models is the optimal balance between the privacy and the utility of models.

This is the consequence of the fact that more privacy generally means more noise, which could lead to the reduction of model performance/accuracy. A more detailed analysis of this challenge is presented in deliverable D2.4, while here as a demonstration of proof of concept the results from the Figure 42 will be shortly described.

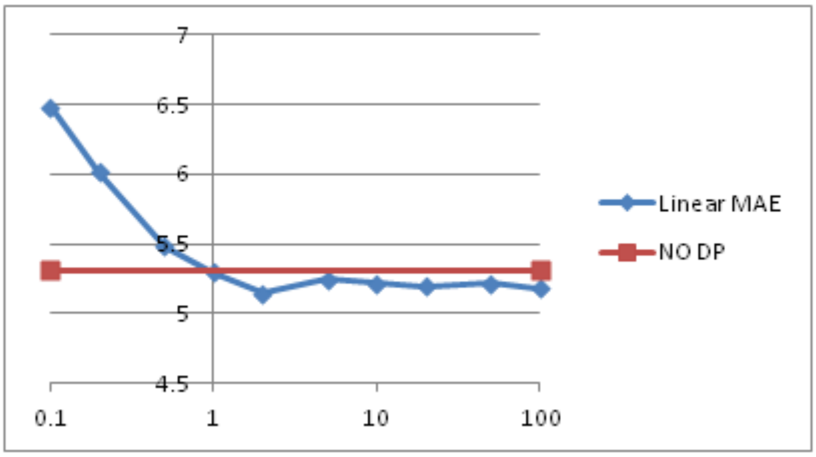


Figure 42. The performance of linear regression model with different values of noise.

Figure 42. shows the Mean Absolute Error (MAE) of the linear regressor for several models built upon datasets with and without noise. Lower values of MAE mean a better regression model. The red line represents the MAE value of linear regression model built upon a dataset without noise. Blue series represent the value of MAE for linear regression models built upon datasets with different values of ε parameter, namely: 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50 and 100. As expected, very low values of ε, which means a lot of added noise, cause a poor performance of regression model (high values of MAE). For the ε=1 the DP model has approximately the same MAE as the model without noise. Surpassingly, for the values ε>1 DP models perform even a little better than model without noise. This and other results suggest that the concept of adding noise to the training data is applicable within ASCAPE project as a privacy preserving technique. Furthermore, the addition of a controlled amount of noise will not have a negative impact on produced machine learning models.

6.3 Homomorphic encryption setup

The core ASCAPE HE services, that are presented in Section 4.3, are enabled through an in-house developed C++ library that offers the necessary functions to encrypt and decrypt the data using the MORE encryption scheme (Table 3).

Homomorphic encryption is special form of encryption that allows data to be encrypted while it is being manipulated. Although a homomorphic cryptosystem is governed by a private key to encrypt and decrypt data, it greatly varies from other forms of encryption as it preserves the algebraic properties of the data to allow a variety of operations to be performed directly on the encrypted data (ciphertext data) without requiring access to the decrypted information (plaintext data) or encryption key. Hence, it represents a promising solution for enabling a third party to process the data in the encrypted form without having to disclose the underlying information.

A variant of a matrix-based homomorphic encryption scheme, called MORE (Matrix Operation for Randomization or Encryption) is employed to enable AI-based data processing on real-data. Following the MORE approach, a matrix-based symmetric secret key is generated upon which a numerical value is encrypted as a matrix and matrix algebra is employed to provide a fully homomorphic behavior. The original message can only be recovered by the owner of the secret key. All operations performed on ciphertext data are matrix-based operations, e.g., addition of plaintext scalars is equivalent to the addition of ciphertext matrices. The considered homomorphic encryption scheme is noise-free (unlimited number of operations can be performed on ciphertext data), non-deterministic (multiple encryptions of the same message and with the same key result in different ciphertexts) and is adapted to directly support floating-point arithmetic. Moreover, it allows a broader spectrum of operations to be performed over encrypted data, including non-linear functions. The 2 by 2 setup of MORE encryption scheme is summarized in Table 3. The order of the regular matrix used to encrypt a message represents a parameter that controls the trade-off between security and efficiency: by increasing the scheme complexity (i.e., matrix order) security is improved at a cost of slightly longer running times.

Table 3. MORE encryption scheme setup over rational numbers.

Message	Scalar value $m \in \mathbb{R}$
Secret key generation	Invertible matrix $S \in \mathbb{R}^{2\times 2}$
Matrix construction	$M = \begin{pmatrix} m & 0 \\ 0 & r \end{pmatrix}$, where $r \in \mathbb{R}$ is a random parameter
Encryption operation	$Encrption(m) = C = SMS^{-1}$
Decryption operation	$Decryption(C) = K = S^{-1}CS$
Message recovery	$m = K_{(1,1)}$

Knowing that ultimately AI models break down to a series of repeating blocks of computations that rely on a limited set of simple operations over floating-point numbers and by leveraging the homomorphic property of the MORE scheme, the functionality of AI models can be extended to hold for operations on ciphertext data. Therefore, the MORE homomorphic encryption scheme enables the use of encrypted data for training and testing AI models, without requiring the content of the decrypted data.

The ASCAPE HE-based machine learning core contains the following main modules:

- `Keygeneration` – a module for generating and storing the MORE encryption key,
- `encryption`- a module for encrypting and storing ciphertext patient data,
- `decryption` – a module for decrypting and storing plaintext patient data.

The `Keygeneration` module is responsible for generating and storing the MORE secret key. The MORE key generation algorithm is shown in Figure 43. The argument needed to call this module is the following:

```
arg1: Path to save the encryption key
```

```
      Output: secret key $\mathbf{S} \in \mathbb{R}^{2\times 2}$
(1) function Keygeneration( )
(2)     while True do
(3)         $\mathbf{S} \longleftarrow \text{RandomUniform}(\text{size} = (2,2), \text{min val}, \text{max val})$
(4)         if $\det(\mathbf{S}) \neq 0$ then \\ Ensure matrix invertibility
(5)             break
(6)         end if
(7)     end while
(8)     return S
(9) end function
```

Figure 43. MORE secret key generation

The `encryption` module is responsible for encrypting the ASCAPE data using the MORE secret key. The encryption algorithm is shown in Figure 44. The arguments needed to call this module are the following:

```
     arg1: Source (plaintext) data file (path +.csv filename)
     arg2: Destination data file to store encrypted values (path
+.csv filename)
     arg3: Path to the generated encryption key
```

```
      Input: Plaintext data $m \in \mathbb{R}$
      Input: Secret key $\mathbf{S} \in \mathbb{R}^{2\times 2}$
      Output: Ciphertext $\mathbf{C} \in \mathbb{R}^{2\times 2}$
(1) function Encryption($m$, S)
(2)     $\mathbf{M} \in \mathbb{R}^{2\times 2} \longleftarrow$ zero matrix
(3)     $\mathbf{M}(\mathbf{0},\mathbf{0}) \longleftarrow m$
(4)     $\mathbf{M}(\mathbf{1},\mathbf{1}) \longleftarrow \text{RandomUniform}(\text{min val}, \text{max val})$
(5)     $\mathbf{C} \longleftarrow \mathbf{S} \times \mathbf{M} \times \mathbf{S}^{-1}$
(6)     return C
(7) end function
```

Figure 44. MORE encryption.

The `decryption` module is responsible for decrypting ciphertext data using the MORE secret key. The decryption algorithm is shown in Figure 45. The arguments needed to call this module are the following:

```
     arg1: Source (ciphertext) data file (path +.csv filename)
     arg2: Destination data file to store decrypted values (path
+.csv filename)
     arg3: Path to the generated encryption key
```

```
    Input: Ciphertext $\mathbf{C} \in \mathbb{R}^{2\times2}$
    Input: Secret key $\mathbf{S} \in \mathbb{R}^{2\times2}$
    Output: Plaintext data $m \in \mathbb{R}$
(1) function Decryption($\mathbf{C}, \mathbf{S}$)
(2)     $\mathbf{K} \longleftarrow \mathbf{S}^{-1} \times \mathbf{C} \times \mathbf{S}$
(3)     $m \longleftarrow \mathbf{K}(\mathbf{0}, \mathbf{0})$
(4)     return $m$
(5) end function
```

Figure 45. MORE decryption.

6.4 Implementation technologies and PoC repositories

The Security Gatekeeper, the component that provides security management, is comprised by the off-the-self components: WSO2 IS and WSO2 APIM. The main components of WSO2 tool are developed with java and in the core are using siddhi and other restfull technologies. WSO2 tool enables easy customisation, configurations and integration changes. The code used for the implementation and the demonstration, is available at the following ASCAPE GitLab repository: *https://gitlab.com/ascape-h2020/wso2-demo.*

The ASCAPE DP component, enabling addition of Laplacian noise on the training data, is implemented in the programming language Python using two free and open-source software libraries: pandas (for data manipulation and analysis) and NumPy (for large, multi-dimensional arrays and matrices). The source code of this component is available at the following ASCAPE GitLab repository: https://gitlab.com/ascape-h2020/machine-learning/a1-missing-value-imputation/-/tree/master/DP

The homomorphic encryption library based on MORE scheme was implemented in C++. This module offers a set of interfaces for encryption, decryption, and key generation. The framework provides C++ helper functions for supporting algebraic operations on private data. It depends on Eigen which is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

7 ASCAPE AI Models Framework

The main focus regarding the cloud framework lied on evaluating and experimenting with different frameworks and selecting the one that best fits the ASCAPE requirements. The second topic of interest was devising a set of proof of concepts that validate the selection.

7.1 Kubeflow as ML Framework

The chosen framework that will serve as the base for the cloud efforts is Kubeflow.

When it comes to machine learning systems, it can be especially challenging to manage the application, platform, and resource considerations. In a cloud environment, ML applications have a different footprint from other web or mobile deployments. For example, a training phase is resource-intensive, while the inference phase is lightweight and speedy. Simultaneously, one needs to find and integrate or develop the tools and frameworks to handle aspects such as configuration, data collection and verification, serving infrastructure, analysis tools, feature extraction, machine learning code, etc. Kubeflow is a framework that contains a curated set of compatible tools specific to ML. It also runs on Kubernetes, the cloud resource orchestration tool agreed with the project partners.

Kubeflow is built around composability, portability, and scalability. Composability allows one to reason about the ML stages as independent systems and use only the parts that make sense for a particular use case (Figure 46.).

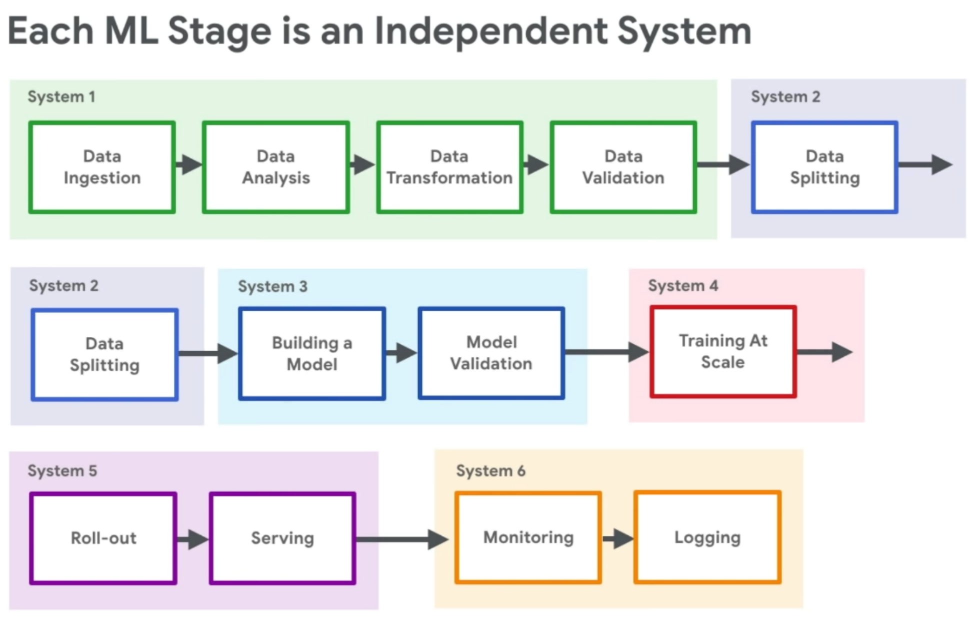


Figure 46. System structuring of ML stages.

Portability allows a researcher to have the same ML workflow experience irrespective of where Kubeflow is running. The same interface and contracts exist whether running locally or in the cloud.

7.1.1 Local installation

In the ASCAPE Gitlab group, we have provided a Vagrantfile that will set up a local instance of Kubeflow for development. The file can be found on the following link: https://gitlab.com/ascape-h2020/cloud/kubeflow-intro/-/blob/master/mini-kf/Vagrantfile. Regardless of the OS, one should be able to use VirtualBox; however, other options like KVM2 on Linux, Hyper-V on Windows, and HyperKit on macOS all work as well.

Kubeflow's pipeline system is built using Python, and having the SDK installed locally will allow one to build pipelines faster. However, if software can't be installed locally, one can still use Kubeflow's Jupyter environment to develop the pipelines.

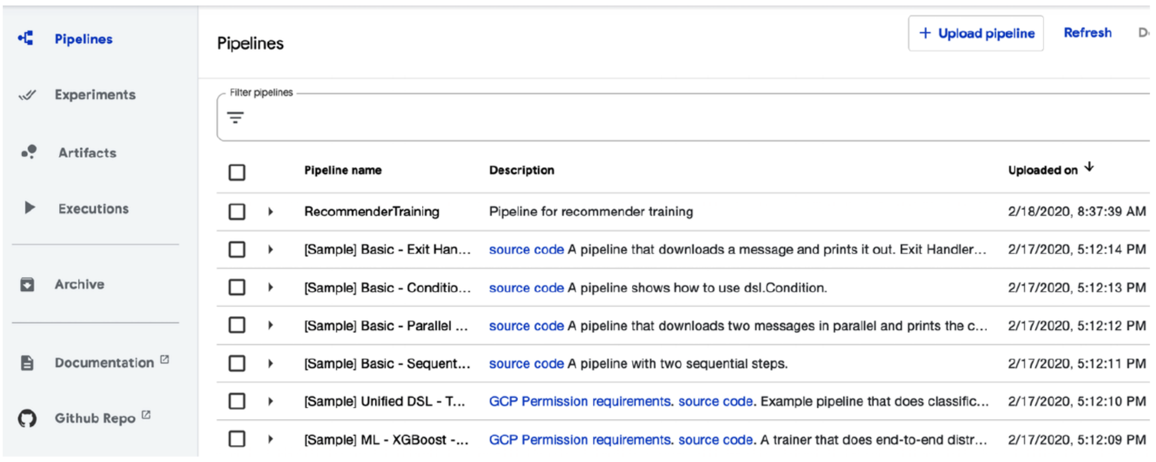
```
virtualenv kfvenv --python python3
source kfvenv/bin/activate
```

Docker is also part of the minimum requirements, allowing one to customize and add libraries and other functionality to the custom containers.

The pipeline system is used to orchestrate ML applications. Orchestration is necessary because a typical ML implementation uses a combination of tools to prepare data, train the model, evaluate performance, and deploy. By formalizing the steps and their sequencing in code, pipelines allow users to formally capture all of the data processing steps, ensuring their reproducibility and auditability, and training and deployment steps.

The Kubeflow Pipelines platform consists of:

- A UI for managing and tracking pipelines and their execution (Figure 47)
- An engine for scheduling a pipeline's execution
- An SDK for defining, building, and deploying pipelines in Python
- Notebook support for using the SDK and pipeline execution



Pipeline name	Description	Uploaded on
RecommenderTraining	Pipeline for recommender training	2/18/2020, 8:37:39 AM
[Sample] Basic - Exit Han...	source code A pipeline that downloads a message and prints it out. Exit Handler...	2/17/2020, 5:12:14 PM
[Sample] Basic - Conditio...	source code A pipeline shows how to use dsl.Condition.	2/17/2020, 5:12:13 PM
[Sample] Basic - Parallel ...	source code A pipeline that downloads two messages in parallel and prints the c...	2/17/2020, 5:12:12 PM
[Sample] Basic - Sequent...	source code A pipeline with two sequential steps.	2/17/2020, 5:12:11 PM
[Sample] Unified DSL - T...	GCP Permission requirements. source code. Example pipeline that does classific...	2/17/2020, 5:12:10 PM
[Sample] ML - XGBoost -...	GCP Permission requirements. source code. A trainer that does end-to-end distr...	2/17/2020, 5:12:09 PM

Figure 47. Kubeflow's Pipelines UI.

7.1.2 Model Training

Containers are used as first-class citizens in Kubernetes. This means that Kubeflow is agnostic to the ML framework that a researcher uses. As mentioned before, training is based on the idea of pipelines and structuring the process in clearly defined steps.

Kubeflow also provides integration between its SDK and tools that are part of the framework. This allows a scientist to use Jupyter notebooks as the source for the training pipelines. The pipeline SDK interaction is abstracted away, allowing the researcher to focus on the ML topic rather than on defining pipelines. Therefore, each notebook cell materializes as a pipeline step (Figure 48).

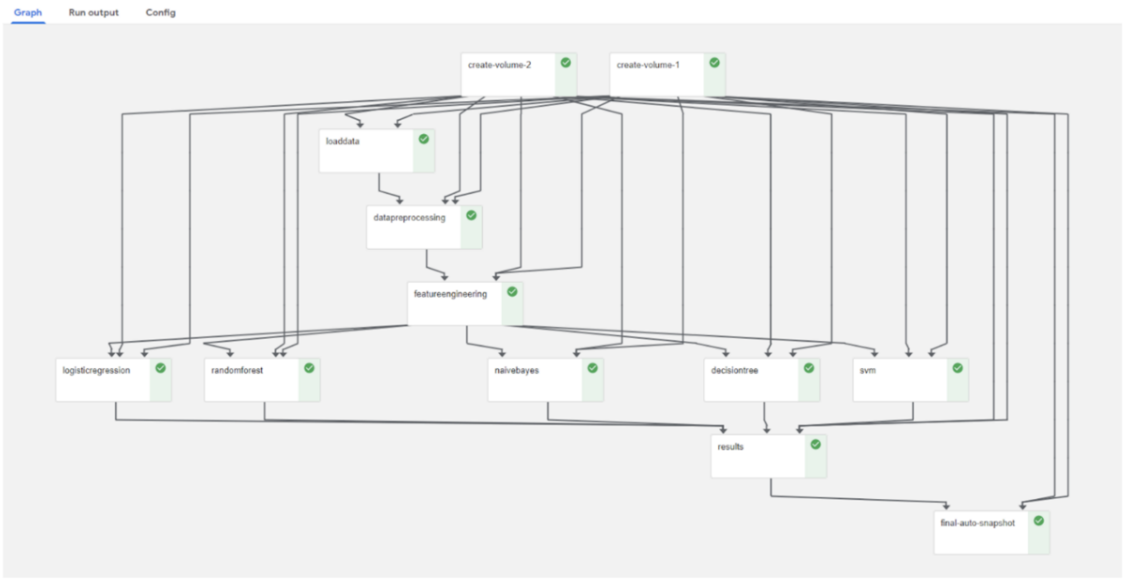
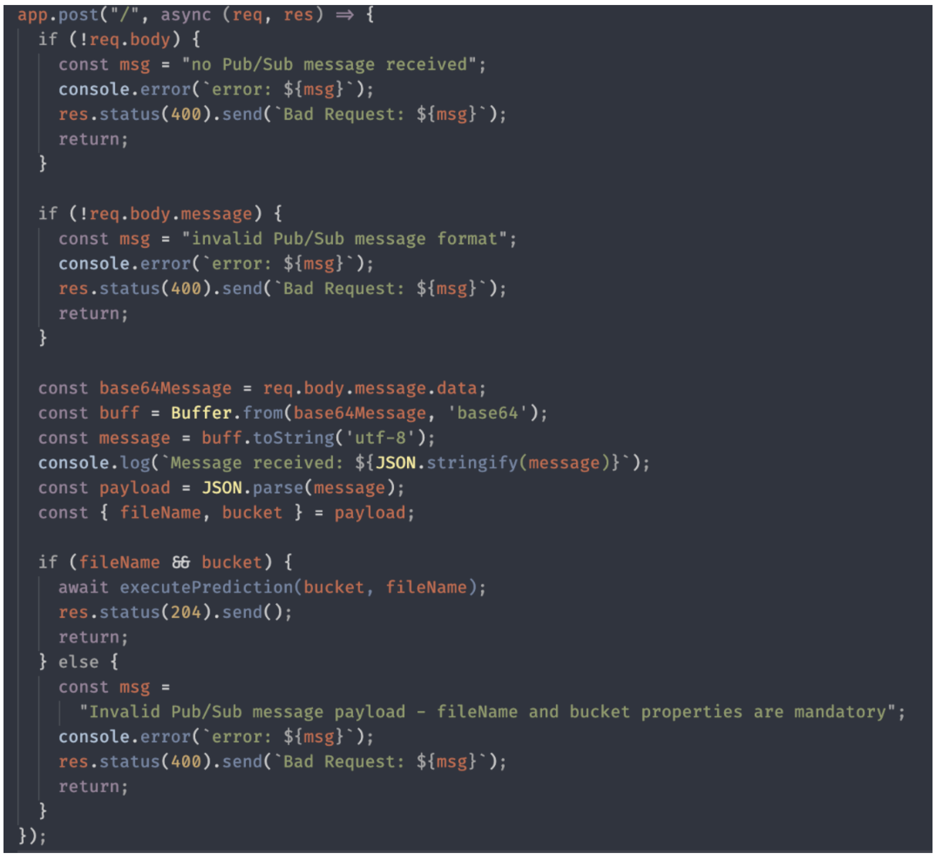


Figure 48. Direct Acyclic Graph of training job.

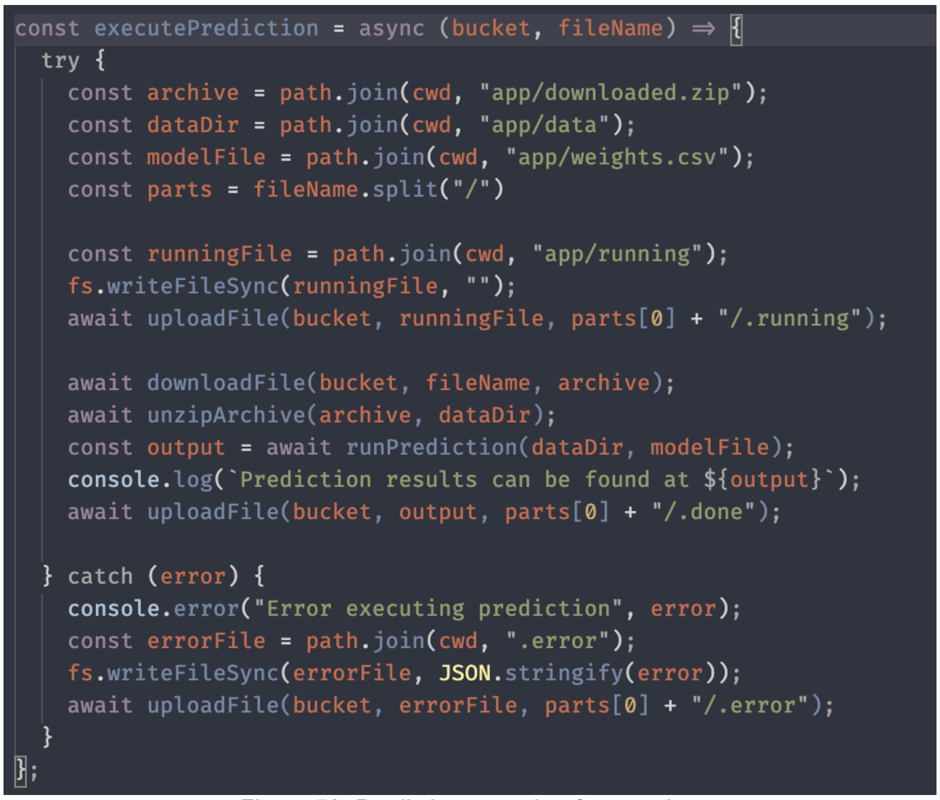
7.1.3 Model Serving

Our research regarding model serving has sought to cover a complete inference solution that includes serving, monitoring, and updating. Serving is responsible for packaging the model in a service that can handle prediction requests. See below part of a containerized web application (Figure 49 and Figure 50) that wraps a homomorphically encrypted model. The full example is committed in the ASCAPE cloud group on Gitlab in the following link: https://gitlab.com/ascape-h2020/cloud/kubeflow-intro/-/tree/master/nodejs-prediction.



```
app.post("/", async (req, res) => {
  if (!req.body) {
    const msg = "no Pub/Sub message received";
    console.error(`error: ${msg}`);
    res.status(400).send(`Bad Request: ${msg}`);
    return;
  }

  if (!req.body.message) {
    const msg = "invalid Pub/Sub message format";
    console.error(`error: ${msg}`);
    res.status(400).send(`Bad Request: ${msg}`);
    return;
  }

  const base64Message = req.body.message.data;
  const buff = Buffer.from(base64Message, 'base64');
  const message = buff.toString('utf-8');
  console.log(`Message received: ${JSON.stringify(message)}`);
  const payload = JSON.parse(message);
  const { fileName, bucket } = payload;

  if (fileName && bucket) {
    await executePrediction(bucket, fileName);
    res.status(204).send();
    return;
  } else {
    const msg =
      "Invalid Pub/Sub message payload - fileName and bucket properties are mandatory";
    console.error(`error: ${msg}`);
    res.status(400).send(`Bad Request: ${msg}`);
    return;
  }
});
```

Figure 49. Web server request handling.



```
const executePrediction = async (bucket, fileName) => {
  try {
    const archive = path.join(cwd, "app/downloaded.zip");
    const dataDir = path.join(cwd, "app/data");
    const modelFile = path.join(cwd, "app/weights.csv");
    const parts = fileName.split("/")

    const runningFile = path.join(cwd, "app/running");
    fs.writeFileSync(runningFile, "");
    await uploadFile(bucket, runningFile, parts[0] + "/.running");

    await downloadFile(bucket, fileName, archive);
    await unzipArchive(archive, dataDir);
    const output = await runPrediction(dataDir, modelFile);
    console.log(`Prediction results can be found at ${output}`);
    await uploadFile(bucket, output, parts[0] + "/.done");

  } catch (error) {
    console.error("Error executing prediction", error);
    const errorFile = path.join(cwd, ".error");
    fs.writeFileSync(errorFile, JSON.stringify(error));
    await uploadFile(bucket, errorFile, parts[0] + "/.error");
  }
};
```

Figure 50. Prediction wrapping & execution.

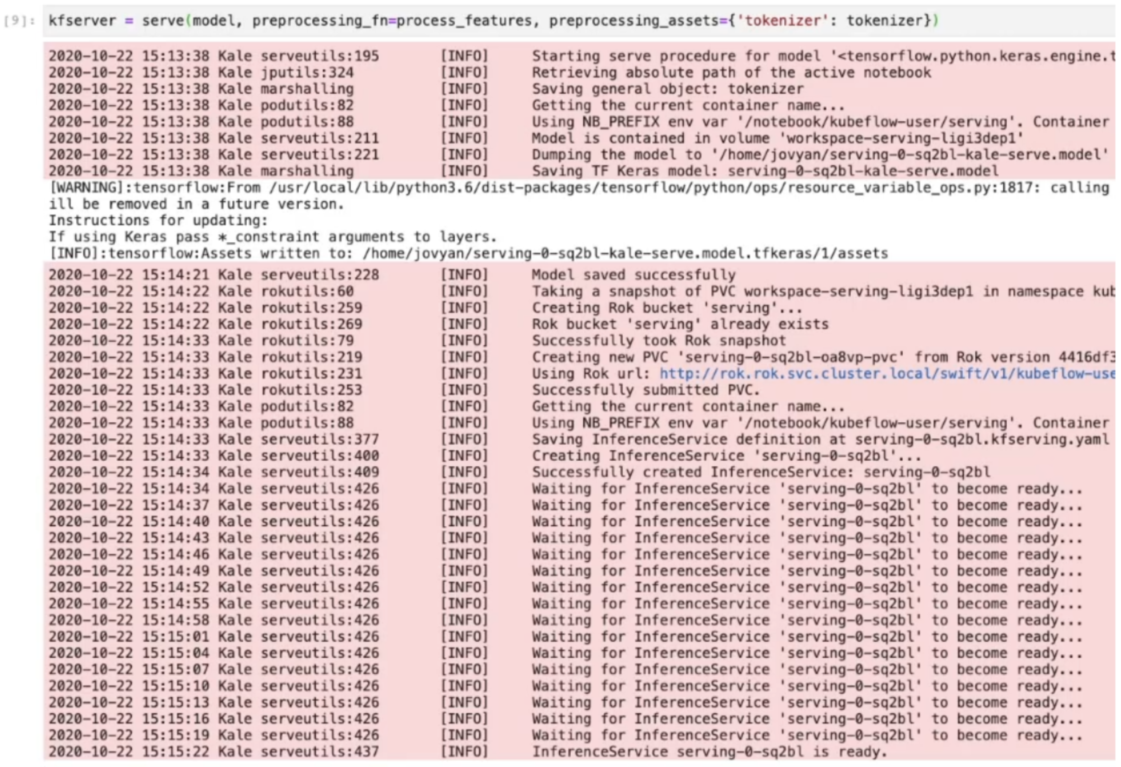
Notice how little needs to be known about the nature of the model being served by the web wrapper. This is because of Docker, and the ability to package images in layers, each responsible for a particular aspect. This means that the trained model is part of an image layer that the web serving component inherits. In the example we have used NodeJS, but swapping to another technology is simple, without any need to intervene in packaging the model.

Model monitoring refers to canning for any irregularities in performance - as well as the underlying model's accuracy. This topic is currently under investigation, as the homomorphic nature of the data makes this area unique.

Model updating fully manages the versioning of the models and simplifies the promotion and rollback between versions.

There are two main approaches for implementing Model as a Service (MaaS): model as code and model as data. Model as code uses model code directly in a service's implementation. Model as data uses a generic implementation that is driven by a model in an intermediate model format like PMML, PFA, ONNX, or TensorFlow's native format. Both approaches are used in different model server implementations in Kubeflow. Our research leads us to believe that when determining which implementation to use, it would be preferable to go with model as data. It allows for the exchange of models between serving instances to be standardized, thus providing portability across systems and the enablement of generic model serving solutions. Most common serving implementations, like TFServing, ONNX Runtime, Triton, and TorchServe, use a model-as-data approach and leverage an intermediate model format. Some of these implementations support only one framework, while others support multiple. Unfortunately, each of these solutions uses different model formats and exposes unique proprietary serving APIs. There is increased friction in swapping between model frameworks, as the interfaces behind these implementations are different. Taking this into consideration, we have chosen Seldon as implementation for model serving. Kubeflow also has good support for it.

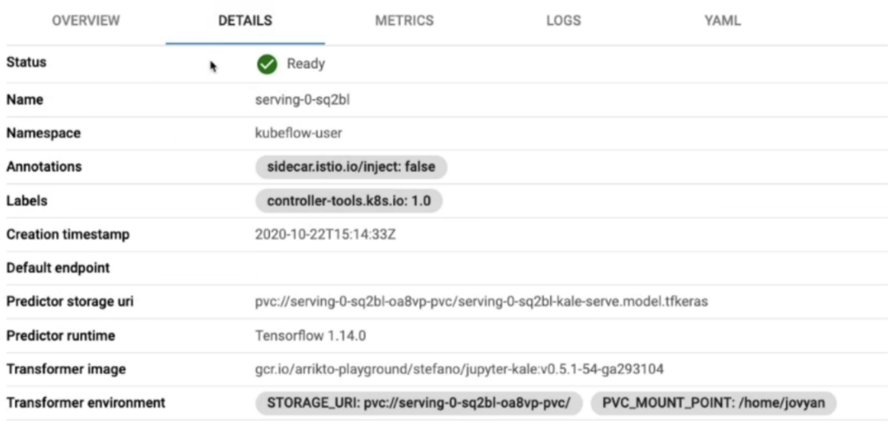
Kubeflow's SDK's composable nature gives the researcher the ability to serve the model straight from a Jupyther Notebook (Figure 51).



```
[9]: kfserver = serve(model, preprocessing_fn=process_features, preprocessing_assets={'tokenizer': tokenizer})

2020-10-22 15:13:38 Kale serveutils:195       [INFO]     Starting serve procedure for model '<tensorflow.python.keras.engine.t
2020-10-22 15:13:38 Kale jputils:324          [INFO]     Retrieving absolute path of the active notebook
2020-10-22 15:13:38 Kale marshalling          [INFO]     Saving general object: tokenizer
2020-10-22 15:13:38 Kale podutils:82          [INFO]     Getting the current container name...
2020-10-22 15:13:38 Kale podutils:88          [INFO]     Using NB_PREFIX env var '/notebook/kubeflow-user/serving'. Container
2020-10-22 15:13:38 Kale serveutils:211       [INFO]     Model is contained in volume 'workspace-serving-ligi3dep1'
2020-10-22 15:13:38 Kale serveutils:221       [INFO]     Dumping the model to '/home/jovyan/serving-0-sq2bl-kale-serve.model'
2020-10-22 15:13:38 Kale marshalling          [INFO]     Saving TF Keras model: serving-0-sq2bl-kale-serve.model
[WARNING]:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/resource_variable_ops.py:1817: calling
ill be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
[INFO]:tensorflow:Assets written to: /home/jovyan/serving-0-sq2bl-kale-serve.model.tfkeras/1/assets
2020-10-22 15:14:21 Kale serveutils:228       [INFO]     Model saved successfully
2020-10-22 15:14:22 Kale rokutils:60          [INFO]     Taking a snapshot of PVC workspace-serving-ligi3dep1 in namespace kub
2020-10-22 15:14:22 Kale rokutils:259         [INFO]     Creating Rok bucket 'serving'...
2020-10-22 15:14:22 Kale rokutils:269         [INFO]     Rok bucket 'serving' already exists
2020-10-22 15:14:33 Kale rokutils:79          [INFO]     Successfully took Rok snapshot
2020-10-22 15:14:33 Kale rokutils:219         [INFO]     Creating new PVC 'serving-0-sq2bl-oa8vp-pvc' from Rok version 4416df3
2020-10-22 15:14:33 Kale rokutils:231         [INFO]     Using Rok url: http://rok.rok.svc.cluster.local/swift/v1/kubeflow-use
2020-10-22 15:14:33 Kale rokutils:253         [INFO]     Successfully submitted PVC.
2020-10-22 15:14:33 Kale podutils:82          [INFO]     Getting the current container name...
2020-10-22 15:14:33 Kale podutils:88          [INFO]     Using NB_PREFIX env var '/notebook/kubeflow-user/serving'. Container
2020-10-22 15:14:33 Kale serveutils:377       [INFO]     Saving InferenceService definition at serving-0-sq2bl.kfserving.yaml
2020-10-22 15:14:33 Kale serveutils:400       [INFO]     Creating InferenceService 'serving-0-sq2bl'...
2020-10-22 15:14:34 Kale serveutils:409       [INFO]     Successfully created InferenceService: serving-0-sq2bl
2020-10-22 15:14:34 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:37 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:40 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:43 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:46 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:49 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:52 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:55 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:14:58 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:01 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:04 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:07 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:10 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:13 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:16 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:19 Kale serveutils:426       [INFO]     Waiting for InferenceService 'serving-0-sq2bl' to become ready...
2020-10-22 15:15:22 Kale serveutils:437       [INFO]     InferenceService serving-0-sq2bl is ready.
```

Figure 51. Snippet for serving a model with KFServing and output logs.

We can then access details about the model configuration parameters and runtime metrics (Figure 52, and Figure 53).



Status	Ready
Name	serving-0-sq2bl
Namespace	kubeflow-user
Annotations	sidecar.istio.io/inject: false
Labels	controller-tools.k8s.io: 1.0
Creation timestamp	2020-10-22T15:14:33Z
Default endpoint	
Predictor storage uri	pvc://serving-0-sq2bl-oa8vp-pvc/serving-0-sq2bl-kale-serve.model.tfkeras
Predictor runtime	Tensorflow 1.14.0
Transformer image	gcr.io/arrikto-playground/stefano/jupyter-kale:v0.5.1-54-ga293104
Transformer environment	STORAGE_URI: pvc://serving-0-sq2bl-oa8vp-pvc/ PVC_MOUNT_POINT: /home/jovyan

Figure 52. Details of Kubernetes service responsible for model serving.

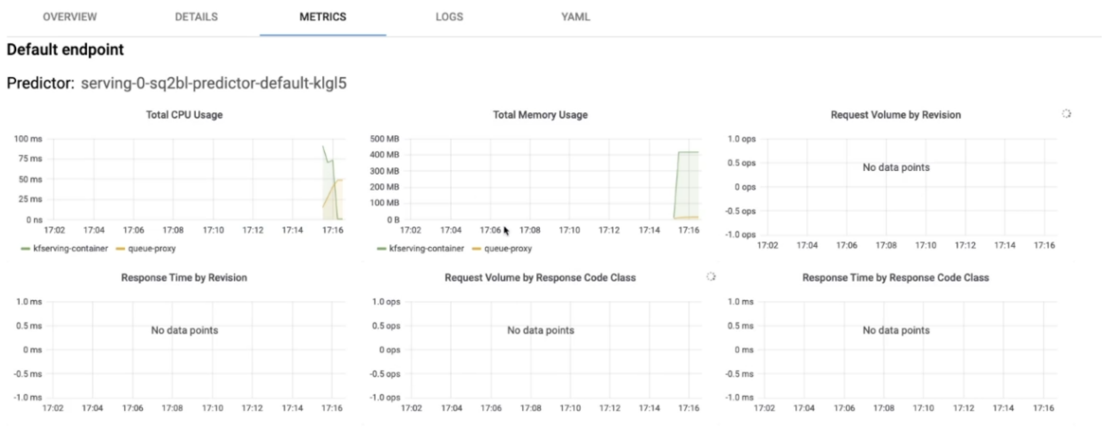

OVERVIEW
DETAILS
METRICS
LOGS
YAML
Default endpoint
Predictor: serving-0-sq2bl-predictor-default-klgl5
Total CPU Usage
Total Memory Usage
Request Volume by Revision
Response Time by Revision
Request Volume by Response Code Class
Response Time by Response Code Class
No data points
kfserving-container
queue-proxy

Figure 53. Parameter monitoring of served model.

7.2 Implementation technologies for PoC demonstration

The components used for the cloud-based model training and serving are based on Kubernetes and open-source operators running on top. For handling the training workflow, we use the Kubeflow project.

We rely mostly on Kubeflow Pipelines for orchestrating machine learning applications. This component allows for structuring data preparation, model training, and the deployment step as a direct acyclic graph. Each node in the graph is written in Python and packaged as a container. The scheduling of the pipeline execution is done using Kubernetes primitives.

To better structure and manage the machine learning metadata, we use MLFlow's Python API.

We use a serverless approach for model serving, built on KNative, called KFServing, that exposes the prediction functionality as an HTTP endpoint.

8 Conclusion

This deliverable reports on current PoC and implementation of different essential components and services of the ASCAPE architecture that are going to be fully implemented in the ASCAPE prototype until the end of the project. The main outcomes of the deliverable are:

- Essential components of the Edge-Cloud Architecture were carefully considered. k3s was used to illustrate the implementation of a Placeholder Container for each component in the ASCAPE Architecture. Deployment configurations for the ASCAPE Edge Nodes and the ASCAPE Cloud are encouraging for further successful continuation of these activities.
- Developed data management services on small subset of fake/anonymized data showed that process of data harmonization, transformations and aggregations from different sources can be successfully standardized. Presented mapping of data to HL7 FHIR format and their codification to SNOMED CT will most likely be applicable on prospectively collected data as well.
- ML and AI techniques implemented to support and illustrate ASCAPE continuous learning, showed to be promising. Comprehensive experiments, performed on 10 datasets derived from retrospective data, exhibited satisfactory results in all presented activities: AI based data pre-processing, ASCAPE federated learning with global and local predictive QoL models, use of homomorphic encryption mechanism to collect encrypted patient data.
- Evaluation of feature attribution and surrogate models (decision tree and linear regression) as mechanisms for explainability showed accuracies that are very similar to the target models. Being rather promising and relaying on initially implemented simulation-based approach, they represent a good starting point for further work on explainable AI within the ASCAPE project. In the future work the clinical partners will have an important role of interpreting and assessing the quality of outputs produced by explainable AI algorithms.
- Mechanisms that ensure security and privacy of the data within the ASCAPE framework, i.e. a security management mechanism, an encryption block and a privacy block, are adequately considered and presented. The justification for the efficient use of one WS02 Identity Server instance and one WS02 API Manager and their deployment in the ASCAPE framework is given. Cloud server installation is a good illustration of the usability of the proposed tool. For privacy preserving based on DP method selection of appropriate value of epsilon parameter influences the accuracy of prediction models. Using different epsilon values the PoC demonstration was focused on one dataset and on several prediction models. The core ASCAPE HE services are illustrated using an in-house developed library that supports functions to encrypt and decrypt the data.
- Selection of Kubeflow framework, that contains a curated set of compatible tools specific to ML, is justified. Kubeflow's pipeline system to orchestrate ML applications with key steps is illustrated as initial PoC. It was shown as appropriate for future full implementation of ASCAPE architecture i.e to serve as the base for the Cloud efforts.

Overall, the experiments, the evaluations, and the initial implementations of important components of the ASCAPE Architecture, are encouraging for future work and for continuation of implementation of the ASCAPE architecture.

9 References

[1] [Online]. Available: https://hapifhir.io/.

[2] [Online]. Available: https://nifi.apache.org/.

[3] [Online]. Available: https://swagger.io/tools/swagger-ui/.

[4] Spring Boot, [Online]. Available: https://spring.io/projects/spring-boot.

[5] MongoDB, [Online]. Available: https://www.mongodb.com/.

[6] MinIO, [Online]. Available: https://min.io/.

[7] ElasticSearch, [Online]. Available: https://www.elastic.co/.

[8] GraphQL Spring Boot, [Online]. Available: https://github.com/graphql-java-kickstart/graphql-spring-boot.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[10] F. Chollet and others, "Keras: The python deep learning library," *Astrophysics Source Code Library,* pp. ascl-1806, 2018.

[11] A. e. al., "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[12] Aaron Parecki et al., "OAuth 2.0", [Online]. Available: https://oauth.net/2 [Accessed 5 January 2021].

[13] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpretable Model Predicitions," NIPS Proceedings, http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions, 2017.

[14] M. A. de Graaf and e. al, "Matching, an appealing method to avoid confounding?," *Nephron Clinical Practice,* pp. 315-318, 2011.

[15] WSO2, "WSO2 - Next generation technologies", [Online]. Available: https://wso2.com [Accessed 5 January 2021].

[16] WSO2, "WSO2 - Identity Server", [Online]. Available: https://wso2.com/identity-and-access-management [Accessed 5 January 2021].

[17] WSO2, "WSO2 - API Manager", [Online]. Available: https://wso2.com/api-management [Accessed 5 January 2021].

[18] The Linux Foundation, "OpenAPI Specification Version 3.0.3", [Online]. Available: http://spec.openapis.org/oas/v3.0.3 [Accessed 5 January 2021].

[19] MongoDB, [Online]. Available: https://www.mongodb.com/.

APENDIX A: List of links to the repositories

Components	**Repository**
API for transformation and channel	• HAPI FHIR repository
ASCAPE Weather Data Adapter	• https://gitlab.ubitech.eu/dsa/ascape/data-workflow-orchestrator • https://gitlab.ubitech.eu/dsa/ascape/data-handler • https://gitlab.ubitech.eu/dsa/ascape/data-parser • https://gitlab.ubitech.eu/dsa/ascape/data-explorer
Fitbit Device Data Adapter	• https://gitlab.ubitech.eu/dsa/ascape/data-workflow-orchestrator • https://gitlab.ubitech.eu/dsa/ascape/data-handler • https://gitlab.ubitech.eu/dsa/ascape/data-parser • https://gitlab.ubitech.eu/dsa/ascape/data-explorer
The source code of ASCAPE core machine learning services	• https://gitlab.com/ascape-h2020/machine-learning/coreedgeml
The source code of the federated learning server and the federated learning client	• https://gitlab.com/ascape-h2020/machine-learning/fl-prototype
The source code for model evolution	• https://gitlab.com/ascape-h2020/machine-learning/a3-model-evaluation/-/tree/master/Homomorphic%20Encryption
A4-Explainability contains the wrapper class for explainable AI, surrogate training and -evaluation	• https://gitlab.com/ascape-h2020/machine-learning/a4-explainability
A5-Simulations contains all the code to run the simulations itself	• https://gitlab.com/ascape-h2020/machine-learning/a4-explainability
Security Gatekeeper	• https://gitlab.com/ascape-h2020/wso2-demo
ASCAPE DP component	• https://gitlab.com/ascape-h2020/machine-learning/a1-missing-value-imputation/-/tree/master/DP