Project Title	Hybrid Eco Responsible Optimized European Solution
Project Acronym	HEROES
Grant Agreement No.	956874
Start Date of Project	01.03.2021
Duration of Project	24 Months
Project Website	heroes-project.eu

D2.1: Definition of CAE Workflow and AI Workflow

Work Package	**WP2, Definition of CAE Workflow and AI Workflow**
Lead Author (Org)	**Ana Gutierrez (HPCNow!)**
Contributing Author(s) (Org)	**Jose E. Torres (HPCNow!)** **Pere Puigdomenech (HPCNow!)** **Philippe Bricard (UCit)**
Reviewed by	**Anaëlle Dambreville (UCit)** **Sablin Amon (UCit)** **Benjamin Depardon (UCit)**
Approved by	**Management Board**
Due Date	**15.08.2021**
Date	**15.09.2021**
Version	**V1.0**

Dissemination Level

X	PU: Public
	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)

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

Versioning and contribution history

Version	Date	Author	Notes
0.1	16.07.2021	Ana Gutierrez (HPCNow!)	V.0
0.2	30.07.2021	Pere Puigdomenech (HPCNow!)	V.1
0.3	10.08.2021	Jose E Torres (HPCNow!)	V.2
0.4	23.08.2021	Anaelle Dambreville (UCit)	Review
0.5	30.08.2021	Jose E Torres (HPCNow!)	Added 4.1 section
0.6	31.08.2021	Jose E Torres (HPCNow!)	Minor edits and corrections
0.7	03.09.2021	Sablin Amon, Benjamin Depardon (UCit)	Review
0.8	07.09.2021	Benjamin Depardon (UCit)	Review
0.9	07.09.2021	Jose E Torres (HPCNow!)	Final version
1.0	15.09.2021	Corentin Lefèvre (Neovia)	Final version approved by the Management Board

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

Table of Contents

List of Figures

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

List of Tables

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

References and Applicable Documents

[1] OpenFOAM Official website, https://openfoam.org/

[2] CATIA, https://www.3ds.com/products-services/catia/

[3] NX, https://www.plm.automation.siemens.com/global/en/products/nx/

[4] Paraview, https://www.paraview.org/

[5] TensorFlow website, https://www.tensorflow.org/

[6] VNC, https://en.wikipedia.org/wiki/Virtual_Network_Computing

[7] COCO (Microsoft Common Objects in Context),

https://paperswithcode.com/dataset/coco

[8] Openvino documentation,

https://docs.openvinotoolkit.org/latest/omz_models_model_ssd_mobilenet_v2_coco.html

[9] COCO, Common Objects in Context, https://cocodataset.org/#home

[10] Microsoft COCO: Common Objects in Context,

https://arxiv.org/pdf/1405.0312.pdf

[11] MobileNetV2, https://tfhub.dev/google/openimages_v4/ssd/mobilenet_v2/1

[12] opencv-python, https://pypi.org/project/opencv-python/

[13] RYAX, https://ryax.tech/

[14] Python, https://python.org

[15] Singularity, https://sylabs.io/singularity/

[16] Singularity Library, https://cloud.sylabs.io/library

[17] Slurm, https://slurm.schedmd.com/

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

Acronyms and Abbreviations

Terminology/Acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
CAD	Computer Aided Design
CAE	Computer-aided engineering
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HPC	High-Performance Computing
IB	InfiniBand
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ML	Machine Learning
OCI	Open Container Initiative
RAM	Random Access Memory
RANS	Reynolds Averaged Navier–Stokes
RGB	Red Green and Blue
STL	Standard Tessellation Language
VM	Virtual Machine
VNC	Virtual Network Computing
YAML	Yet Another Markup Language

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

Executive Summary

The HEROES project is aiming at developing an innovative European software solution allowing industrial and scientific user communities to easily submit complex Simulation and ML (Machine Learning) workflows to HPC (High Performance Computing) Data Centres and Cloud Infrastructures. It will allow them to take informed decisions and select the best platform to achieve their goals on time, within budget and with the best energy efficiency.

This document presents the definition of CAE and AI Workflows selected as examples for the project. AI/HPC Workflow integration: the process of containerizing, optimizing, and integrating new applications and workflows in HEROES.

The other tasks of the HEROES project rely on the workflows provided by this Work Package as examples to provide an end-to-end prototype at the end of the project. Existing and new workflows will be designed, deployed, and operated to allow AI/HPC users to focus on their daily work.

For each of the workflows provided, a set of requirements have been defined and developed as follows:

- Definition
- High-level diagram and set of reproducible steps
- Details of implementation and requirements

The development of this document and steps is closely linked to the optimisation work of WP3.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

1 Introduction

The objective of this document is to understand the applications and workflows that will be taken as examples in the HEROES project to integrate in the AI/HPC as a service platform, as defined on the WP2 (AI and HPC Workflows).

Science-based model is being combined with simulations, big data, and AI/ML to solve complex problems and phenomena. The growth of data at large scale has proven to be a paradigm shifter for computational science. Scientific computational problems have been faced with the need to analyse increasing amounts of data as part of their application workflows.

Workflows are composed by a set of reproducible steps that can be executed as a pipeline to obtain a desired result, saving iteration time by reducing debugging steps and deployment time.

The goal is to enable analysis by bringing a user-friendly approach to complex engineering issues with shareable interfaces. The flexibility of the available tools will provide the possibility to apply this methodology to other types of simulation in a short time.

This will lead to the development of a methodology to easily define the workflows, and the way they can be integrated. Prototypes of the tools and user-interfaces will be developed within the project.

For the workflow implementation we will use:

- Computational Resources: HPC Clusters based on many-core processors and/or accelerated with GPU, also with a high-performance parallel filesystem for AI and HPC workflows.
- HEROES software platform: We will use the HEROES platform to manage heterogeneous HPC infrastructures in scientific and engineering environments. This new platform, combining high performance computing, high performance storage and application containerization, represents a generic technology that can be applied to many other generic workflows.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

2 CAE Workflow

2.1 *Introduction*

The numerical simulation of the Computational Fluid Dynamics is a very well-known strategy to optimize engineering designs in multiple fields inside Computational Aided Engineering.

Both the aerodynamical and the thermal behaviour can be predicted, leading to an optimal manufacturing design.

High Performance Computing (HPC) simulation tools are capable to run these simulation routines in a time compatible with the time to market of any product developed by manufacturing SMEs or big companies.

In this regard, HPC simulation tools can be seen as enabling technologies for the use of CFD and increasing the market share of any related product.

In particular, for any CFD model several steps are needed to be accomplish one after the other, before obtaining the final result, some of these steps are requiring manual interaction of the user. The objective of HEROES is to help CFD users to accomplish the final result in an easier way, while optimising the HPC platform and tools.

The ultimate objectives are:

- To reduce the human interaction by using automation during manual steps
- To optimise the use of HPC resources
- To reduce the time needed to accomplish one CFD workflow

2.2 *Definition of the CFD workflow*

The main objective of the workflow automation is to reduce the human interaction with the HPC system and maximise automation in each step.

Before starting the selected CAE workflow, a requirement must be considered, the geometry that will be used as an input for the CAE workflow must meet certain conditions.

First, a preliminary geometry treatment must be done to obtain the file to be analysed. Then, the 3D CAD model must be simplified to remove irrelevant details that lead to an excessive number of cells. This operation, hereafter called CAD preparation, is performed manually with CAD software as CATIA [2] or NX [3].

In Figure 1, the whole CAE workflow is illustrated with all the steps that must be automated to accomplish the CFD analysis.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

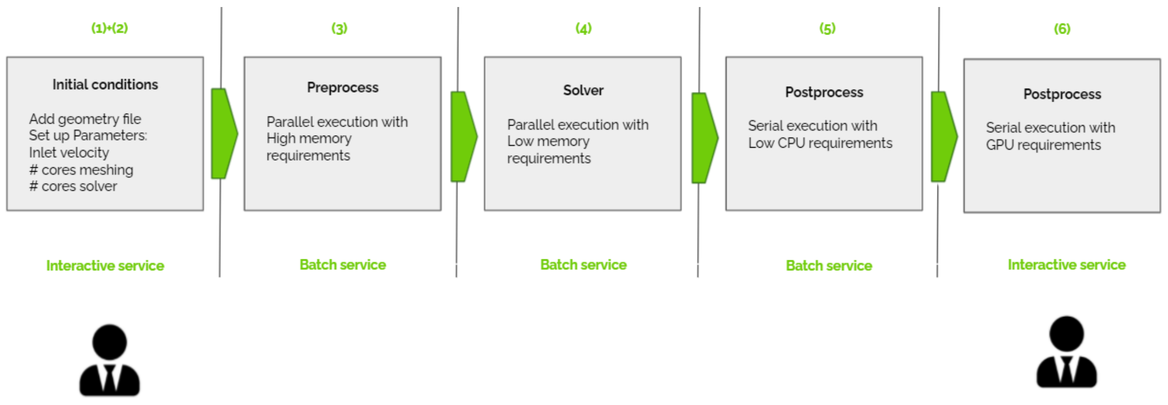


Figure 1. High Level Diagram of the CAE Workflow

The steps are:

(1) + (2) Initial conditions: During this step, two actions will be done: add the geometry file and set up of parameters. This includes, for instance, to assign boundary conditions or select the solver procedure. Those tasks are normally done manually. As a result, launching time becomes quicker and reduces the risk of human mistakes.

(3) Pre-process: This step converts the given geometry into a convenient mesh. A complex multipart-multi-domain mesh is always necessary due to the complexity of the model and the different materials used. The lack of automatic algorithms to assemble and manage multiple meshes requires an excessive manual modelling time and increases risks of human errors. With the CAE Workflow, all the meshing process will be automated using OpenFoam [1] and bash scripts. The refinement process will need a big amount of memory to convert the geometry into a proper meshed domain.

(4) Solver: Consists of the resolution of the CFD model, according to the previously defined parameters. This step will use most of the compute power.

(5) Post-process: Some of the post-process tasks can be automated using Paraview [4] macros or OpenFoam functions. Because of the lack of customized post-processing tools (such as cutting planes or x-y plots), excessive time for verification and post-processing is required. There is also a need to manage the files obtained from the CFD solver, as this kind of file tends to be very big.

(6) Post-process: Sometimes additional post-process tasks can be required after the whole CAE workflow has been executed. For this reason, a final step can be executed allowing the user to interact through a remote desktop with the output of the whole CAE workflow execution.

The main drawbacks of manual CAE workflow process are the excessive working hours, the high cost in license fees and the limited number of simulation rounds. The whole process takes between 3-4 weeks depending on the complexity of the CFD model.

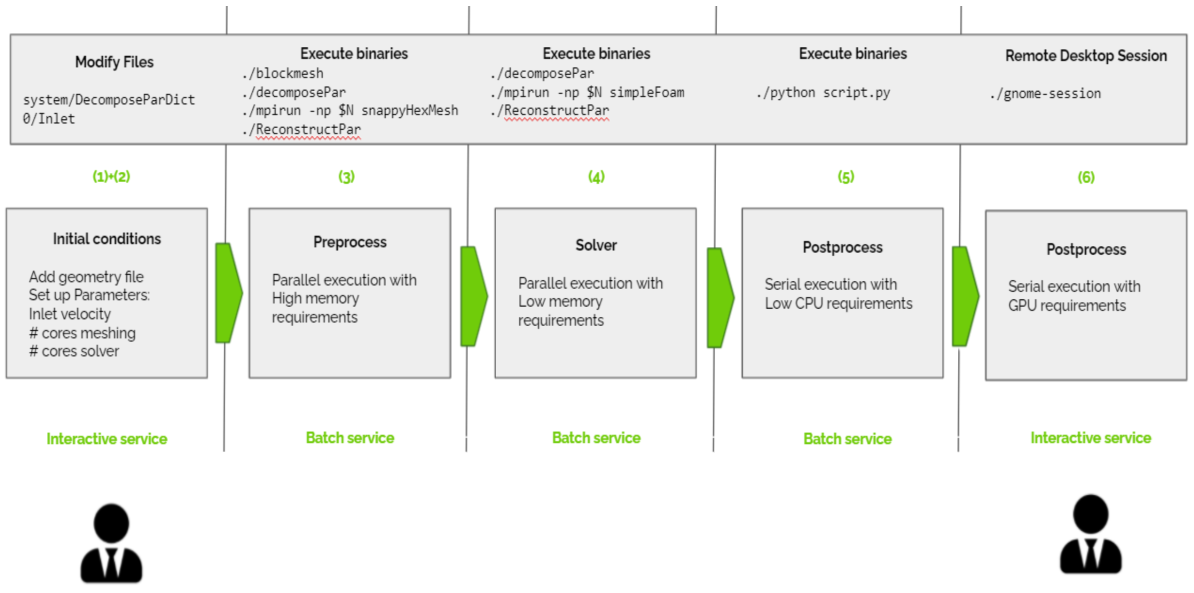
The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

2.3 *Workflow High Level Diagram*

All the steps of the CAE workflow will be executed on the HEROES platform. Each step will be sent to a specific computing architecture depending on the resources consumption in order to optimize the efficiency of the whole workflow.

Some parts of the workflow will require an HPC infrastructure and other simple cloud instances.

All the CFD models will be computed in an HPC as a service infrastructure, removing the maintenance costs of the local systems and providing access to scalable HPC systems which provide better performance.



```
system/DecomposeParDict
0/Inlet
```

```
./blockmesh
./decomposePar
./mpirun -np $N snappyHexMesh
./ReconstructPar
```

```
./decomposePar
./mpirun -np $N simpleFoam
./ReconstructPar
```

```
./python script.py
```

```
./gnome-session
```

Figure 2. CAE Workflow with details of the implementation

2.4 *Details of the implementation*

The chosen model for the CAE workflow is a classic CFD simulation of the airflow around a motorbike (Figure 3).

It is a popular OpenFoam example for the SimpleFoam solver. SimpleFoam is a steady-state solver for incompressible flows with turbulence modelling which is used to solve Navier-Stokes equations with a RANS approximation. With these results the different aerodynamic coefficients like drag or lift can be determined.

The size of the example model has been increased from approx. 200,000 to approx. 40,000,000 elements, modifying blockMeshDict and snappyHexMeshDict files, to run a medium-sized real use case.

```
Input name: motorBike
Number of elements: 41,631,282
OpenFOAM version: v1812
```

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

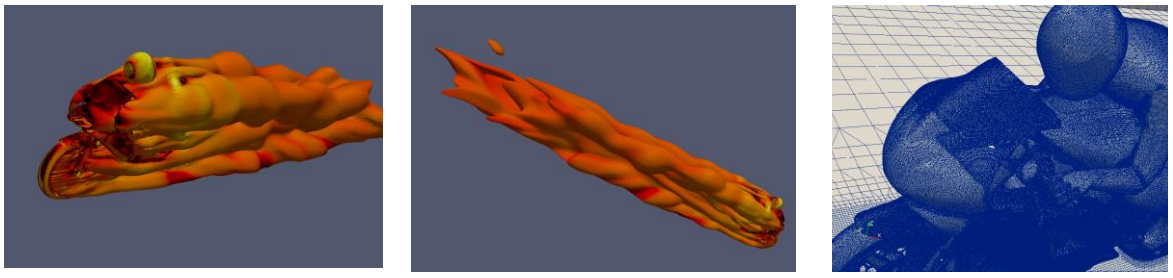


Figure 3. Graphical representation of the post-processed images generated with the Motorbike model

All stages of the process are carried out in the computing infrastructure using containerization in all the steps by means of the HEROES platform except for the CAD preparation stage. The main stages of the process are (Figure 2):

(1) + (2) Initial conditions: During this first step, the user will have to take two actions using the platform interactive service:

1. Upload geometry file

   The 3D representation of the object (STL file) that will be studied will be uploaded to the Data Management module (see Deliverable 3.1). The user will just need to choose the file inside its device and the file will be automatically uploaded to the HEROES platform.

2. Setup initial conditions

   The users will be allowed to set up three parameters for the whole CAE workflow: speed of the airflow of the inlet [m/s] (inlet velocity), cores for the meshing process and cores for the solver. Then the HEROES platform will call OpenFoam to automatically create domains and modify parameters in the input file. The automation of this step consists of a combination of OpenFoam and bash scripts to parse and modify the following files:

```
system/DecomposeParDict
0/include/InitialConditions
```

The computing requirements needed for the execution of this step are:

Table 1. Computing Requirements Initial Setup Conditions

(1) + (2) Initial conditions	Requirements
CPU	2 cores
MEM	Some GB
DISK	200 MB
EXECUTION TIME	A few seconds

For this step a light-weight computational node can be used.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

After this interactive service is accomplished, all the following steps will be executed as batch services and no human interaction will be required anymore.

(3) Pre-process: During this step, the CFD model resolution starts with a pre-process procedure that will consist of:

1.	Execute primary mesh with a serial binary	`blockMesh`
2.	Decompose the primary mesh with scotch method	`decomposePar`
3.	Execute mesh refinement with a parallel binary	`snappyHexMesh`
4.	Reconstruction of the mesh	`ReconstructPar`

The use of the SnappyHexMesh procedure of OpenFoam [1] allows to automatically mesh all domains, cutting off hours of manual work. In addition, parallelization is an important advantage when big meshes are needed.

The computing requirements needed for the execution of this step are:

Table 2. Computing Requirements Pre-process Step

(3) Pre-process	Requirements
CPU	32 cores
GPU	N/A
MEM	12 GB/core
DISK	5 GB
EXECUTION TIME	~4 hours

For this step a computational node with a high volume of RAM (fat node) is needed.

(4) Solver: During this step, the system will require to iterate until finding a solution for the CFD model.

The execution steps that will be required are:

1.	Decompose the refined mesh with a scotch method	`decomposePar`
2.	Execute simpleFoam solver with a parallel binary	`simpleFoam`
3.	Iterate until the convergence condition is reached	
4.	Reconstruct the model	`ReconstructPar`

The computing requirements needed for the execution of this step are:

Table 3. Computing Requirements Solver Step

(4) Solver	Requirements
CPU	256 cores
GPU	N/A
MEM	4 GB/core
DISK	200 GB
EXECUTION TIME	~8 hours

For this step several HPC nodes are needed, a low latency network is a must for an optimal performance.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

(5) Post-process: This is the final step that will run as a batch service, some actions are needed to show to the user the result in a readable way. Paraview is the tool chosen for post-processing. It is fully compatible with OpenFoam [1]. Automatic tools have been developed to create standard post-processes as cutting planes and x-y plots.

The execution steps will be:

1. Execute a python script including a Paraview macro
2. Store image files

The computing requirements needed for the execution of this step are:

Table 4. Computing Requirements Post-Process Step

(5) Postprocess	Requirements
CPU	2 cores
GPU	N/A
MEM	Some GB
DISK	200 GB
EXECUTION TIME	Some seconds

For this step a light-weight computational node can be used.

(6) Post-process: If additional manual work needs to be done by the user, an optional step can be executed. A graphical remote session should be launched. The HEROES platform will expose a Paraview desktop session to allow the user to interact with the solved CFD model. Remote post-processing (using remote visualization tools) avoids waiting time to transfer large files.

The steps will be:

1. Initialize a gnome desktop in the GPU node
2. Initialize a VNC [6] server in the GPU node
3. Expose the desktop session to the user

The computing requirements needed for the execution of this step are:

Table 5. Computing Requirements Post-Process Step

(6) Postprocess	Requirements
CPU	4 cores
GPU	Visualisation GPU
MEM	4 GB/core
DISK	200 GB
EXECUTION TIME	N/A

For this step a GPU node is needed.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

3 AI/ML Workflow

3.1 Introduction

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions.

The ideal characteristic of AI is its ability to rationalize and take actions that have the best chance of achieving a specific goal.

Machine learning (ML) is a subfield of artificial intelligence. The goal of ML is to make computers learn from the data that an end user gives them and adapt to new data without being assisted by humans. The resulting program, consisting of the algorithm and associated learned parameters, is called a trained model.

Within ML, Deep learning techniques enable this automatic learning through the absorption of huge amounts of unstructured data such as text, images, or video.

The goal of this section is to define a complete workflow, composed by a set of reproducible steps, to deploy a functional model that solves a common problem on the AI field.

With the definition of the workflow, we obtain an automated process that simplifies the usage of machine learning models for industrial and scientific communities.

3.2 Definition of the ML workflow

The machine learning workflow is composed by the typical phases implemented during a machine learning project (see Figure 4):

(1) Use case conception and problem formulation: This step consists of the definition of the valuable information to get out of the model.

(2) Data preparation: The potential usefulness and accuracy of the model will depend on the quality of the data gathered during this step. Tasks to be performed will include identifying sources of data, aggregating, and cleaning them into a single dataset.

(3) Features engineering and selection: This step covers the understanding and enhancement of the data. Adding new features to a dataset is a key step in the machine learning process.

(4) Dataset split: This phase involves breaking processed data into three datasets—training, validating, and testing:

1. Training dataset—used to initially train the algorithm and teach it how to process information.
2. Validation dataset—used to estimate the accuracy of the model during the training process.
3. Testing dataset—used to compare the accuracy and performance of the models. This set is also meant to expose any issues that might have appeared during training.

(5) Model building and tuning: Choose the best model suitable to solve the user's problem and define their parameters.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

(6) Model training and testing: This step involves feeding the training dataset to the model. It will then be able to learn appropriate parameters from the features.

(7) Model evaluation: This allows the data scientist to decide if the performance of the model is good enough (proceed to the deployment step) or not (back to step 5 or 3).

(8) Model deployment: This step involves the execution of the model in a compute resource, in which the algorithm runs and is published to work with it on a trained and functional state.

(9) Predictions monitoring and visualization (online evaluation): This step involves the monitorization of the running module to obtain metrics from their execution and behaviour. Tasks will also include re-feeding the model with more accurate data (back to task 2) or re-tune the model (back to task 5) based on its performance.

(10) Models and versions management: The models could also be versioned, allowing their predictions by training date, level of accuracy, etc.

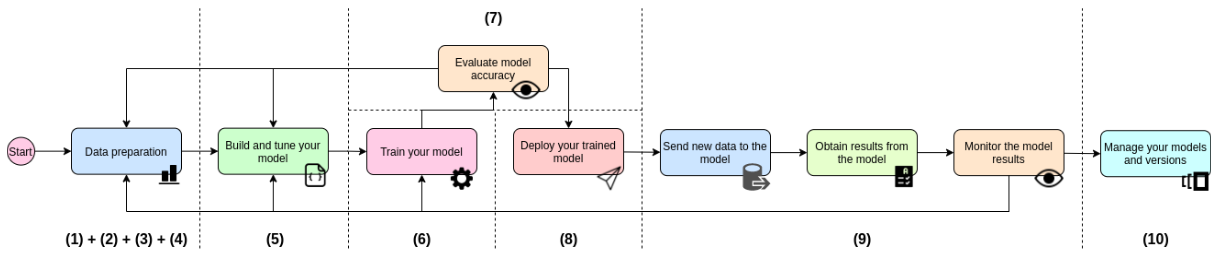

(7)
Evaluate model accuracy
Start
Data preparation
Build and tune your model
Train your model
Deploy your trained model
Send new data to the model
Obtain results from the model
Monitor the model results
Manage your models and versions
(1) + (2) + (3) + (4)
(5)
(6)
(8)
(9)
(10)

Figure 4. Complete Machine Learning workflow

Developing a machine learning model is a process of experimentation and incremental adjustment. Refinement and adjustment play a key role on the process to obtain the best possible results. A threshold of success must be established beforehand.

This process allows the end user to automate some aspects of the machine learning operations workflow, such as training the model and feature selection phases, but probably not all.

3.3 *Workflow High Level Diagram*

All the steps of the ML workflow will be executed on the HEROES platform. Each step will be sent to a specific computing architecture, module, or resource, depending on the usage and resource consumption in order to optimize the efficiency of the whole workflow.

The goal of the chosen ML workflow is to detect different types of objects inside a video file or stream (Figure 5), by splitting it on individual images and applying object identification on each of the obtained frames afterwards.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

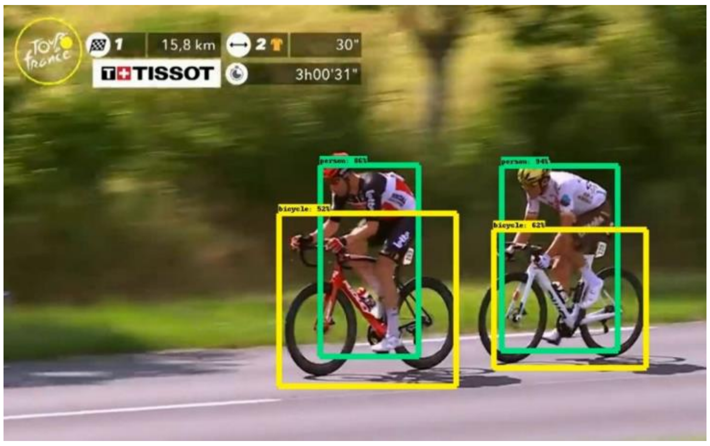


Figure 5. Example of object detection over a video frame

Some parts of the workflows will require the execution of containerized software over HPC infrastructure and other parts will require the execution of containerized software over simpler computational resources.

The chosen model for the ML workflow is a classic object detection example, using a pre-trained model. Some steps of the complete Machine Learning workflow described above are avoided since the initial steps are already built-in (Data Preparation + Training, steps 1 to 7 on Figure 5).

Figure 6 shows the steps of the ML workflow going from the selection of the pre-trained model (1) to the download of the final tagged video (8).

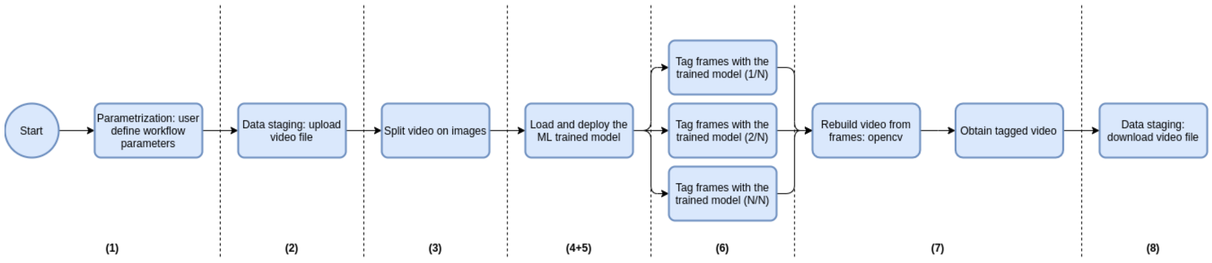

Start
Parametrization: user define workflow parameters
Data staging: upload video file
Split video on images
Load and deploy the ML trained model
Tag frames with the trained model (1/N)
Tag frames with the trained model (2/N)
Tag frames with the trained model (N/N)
Rebuild video from frames: opencv
Obtain tagged video
Data staging: download video file
(1)
(2)
(3)
(4+5)
(6)
(7)
(8)

Figure 6. Diagram for the video object detection ML workflow inside HEROES platform

3.4 *Details of the implementation*

TensorFlow [5] is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

The chosen pre-trained classification model (ssd_mobilenet_v2_coco) [7, 9, 10] is a Single-Shot multi-box Detection (SSD) network (MobileNetV2) [11] implemented on the TensorFlow [5] framework (part of the TensorFlow object detection API) [5]. It intends to perform object

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

detection, going straight from image pixels to bounding box coordinates and class probabilities on detected objects.

This model has been trained on the Microsoft Common Objects in Context (MS COCO) [10] image dataset. The MS COCO [7, 9, 10] dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images of 91 object types, with a total amount of data of 18GB. In the dataset, the objects are labelled using per-instance segmentations to aid in precise object localization.

The model input is a single image in RGB format while the outputs are the detected objects (classes) and their locations (bounding boxes) in the image.

The steps that the model performs internally are:

- Obtain the detected bounding boxes with their corresponding classes and confidences for a given image
- Inspect which classes are detected
- Visualize each detection
- Visualize all the boxes predicted by the net scaled by their "objectless" measures
- Visualize all the boxes scaled by the probability that they contain an object

The object detection workflow proposed for the HEROES platform follows these steps as represented on Figure 6:

(1) Initial conditions (Parametrization): During this step, an interactive service will be implemented, and the user will have to determine the parameters to use in the workflow.

1. Select a pre-trained ML model for computer vision and object identification from a catalogue (the ssd_mobilenet_v2_coco pre-trained ML model in our case) [7, 9, 10].
2. Select additional parameters that can affect the ML model behaviour.

(2) Data staging: During this step the user selects a video file to be processed by the object detection workflow. The file is uploaded to the data store used on the HEROES platform thanks to the Data Management module.

(3) Data preparation: During this step, the uploaded video is pre-processed. This step executes a Python script using the library opencv-python [12] that cuts a video into frames and returns a directory that contains individual frames. An example of such code is shown in Figure 7.

```
# Find video filename
      filename = req["video"]
      cap = cv2.VideoCapture(filename)
      idx = 0

# Read video and save each frame
      while True:
            ret, frame = cap.read()
            if not ret:
              break
            image_path = os.path.join(output_dir + "/" + str(idx).zfill(10)
+ ".jpg")
            cv2.imwrite(image_path, frame)
```

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

```
            idx += 1

# Saving video infos

      save_video_info(cap, output_dir, filename, idx)
      return {"frames": output_dir}
```

Figure 7. Example Python code for open a video file and split it into frames using opencv

All frames will be numbered and a JSON file will be created with some information about the video (Filename, FPS, FourCC, Frames, ...).

The JSON file should contain information such as:

```
{"fps": 25.0, "width": 1920, "height": 1080, "fourcc": 828601953,
"frame_count": 393, "ext": ".mp4", "filename": "Video"}
```

The metadata inserted in the JSON file works like an index, allowing the reconstruction of the video at a later stage.

The computing requirements needed for the execution of this step are based on the video splitting process. This step could be quite intensive depending on the size of the file.

Table 6. Computational Requirements for Data preparation step

(3) Data Preparation step	Requirements
CPU	16 cores
GPU	N/A
MEM	1 GB/core
DISK	5 GB
EXECUTION TIME	~1 hour

(4) Load ML model: Loads the trained ML model selected by the user during the workflow definition. This step uses custom Python code to pull a pre-trained model from an external repository or registry, allowing to select and use a huge collection of models and versions. An example of such code is shown in Figure 8.

```
# Downloading model
      MODEL_DATA = model_name + ".tar.gz"
      print(f"Dowloading model {MODEL_URL + MODEL_DATA}")
      r = http.request("GET", MODEL_URL + MODEL_DATA)
      if r.status == 200:
            TF_MODEL_PATH = TMP_DIR + "/" + MODEL_DATA
            f = open(TF_MODEL_PATH, "wb")
            f.write(r.data)
            f.close()
            tar_file = tarfile.open(TF_MODEL_PATH)
            for file in tar_file.getmembers():
                  file_name = os.path.basename(file.name)
```

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

```
                    if "frozen_inference_graph.pb" in file_name:
                          tar_file.extract(file, TMP_DIR)
                          print(f"Extracted file {file.name} on directory 
{TMP_DIR}")
                          os.remove(TF_MODEL_PATH)
```

Figure 8. Example Python code for download a ML model from an URL and extract it locally

(5) Model deployment: The computing requirements needed for the execution of this step are based on deploying the Tensorflow[5] ML model containerized inside a compute resource, like a VM, a dedicated machine or a cluster.

(6) Asking for predictions: The process uses the deployed model to analyse the image frames obtained on the previous steps and tagging the objects detected on each of them. This step is recommended to be run in parallel to obtain a better speedup on the tagging and image processing job. An example of such code is shown in Figure 9.

```
# Start detection for each image
      for file_name in imagesList:
            img = cv2.imread(file_name)
            output_dict = run_inference_for_single_image(
                  np.expand_dims(img, axis=0),
                  sess,
                  tensor_dict,
                  detection_masks,
                  detection_boxes,
            )

            # Draw boxes
            vis_util.visualize_boxes_and_labels_on_image_array(
                  img,
                  output_dict["detection_boxes"],
                  output_dict["detection_classes"],
                  output_dict["detection_scores"],
                  category_index,
                  instance_masks=output_dict.get("detection_masks"),
                  use_normalized_coordinates=True,
                  line_thickness=8,
            )

            # Save tagged image
            _, name = os.path.split(file_name)
            name, ext = os.path.splitext(name)
            tagged_image = f"{output_dir}/{name}_tagged{ext}"
            cv2.imwrite(tagged_image, img)
```

Figure 9. Example Python code to start the object detection on each frame

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

Table 7. Computational Requirements for Asking for predictions step

(6) Asking for predictions	Requirements
CPU	32 cores
GPU	N/A
MEM	12 GB/core
DISK	5 GB
EXECUTION TIME	~2 hour

(7) Post-processing: This step will rebuild the initial video file using the tagged frames generated by the ML model. To achieve the goal, this step executes a Python script to rebuild the video file from the frames. Depending on the number of frames to process, this could result in an intense compute resource usage.

To write the video, this step needs the numbered frames and the JSON file containing the video and frame information. An example of such code is shown in Figure 10.

```
# Prepare video writer
      out, video_path = prepare_output(info, output_path)
      print(f"Writing tagged video to {video_path}!")
      for f in frameList:
            # Write video
            image_np = cv2.imread(f)
            out.write(image_np)
      out.release()
```

Figure 10. Example Python code to start the video rebuild from frames

Table 8. Computational Requirements for Post-processing step

(7) Post-processing	Requirements
CPU	16 cores
GPU	N/A
MEM	6 GB/core
DISK	5 GB
EXECUTION TIME	~1 hour

(8) Data staging: This step relocates the obtained tagged video to a data storage where the user can download the resulting file from the workflow.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

4 Workflow definition and integration on HEROES

4.1 *Workflow definition and implementation*

The integration of these workflows within the HEROES platform is accomplished using an internal architecture module in charge of developing, orchestrating, maintaining, and deploying the workflows on the available computational resources.

The Workflow and Job Management module is based on a third-party framework called RYAX [13]. This provides some functionalities to simplify the way to create, manage and deploy advanced workflows through a GUI or API.

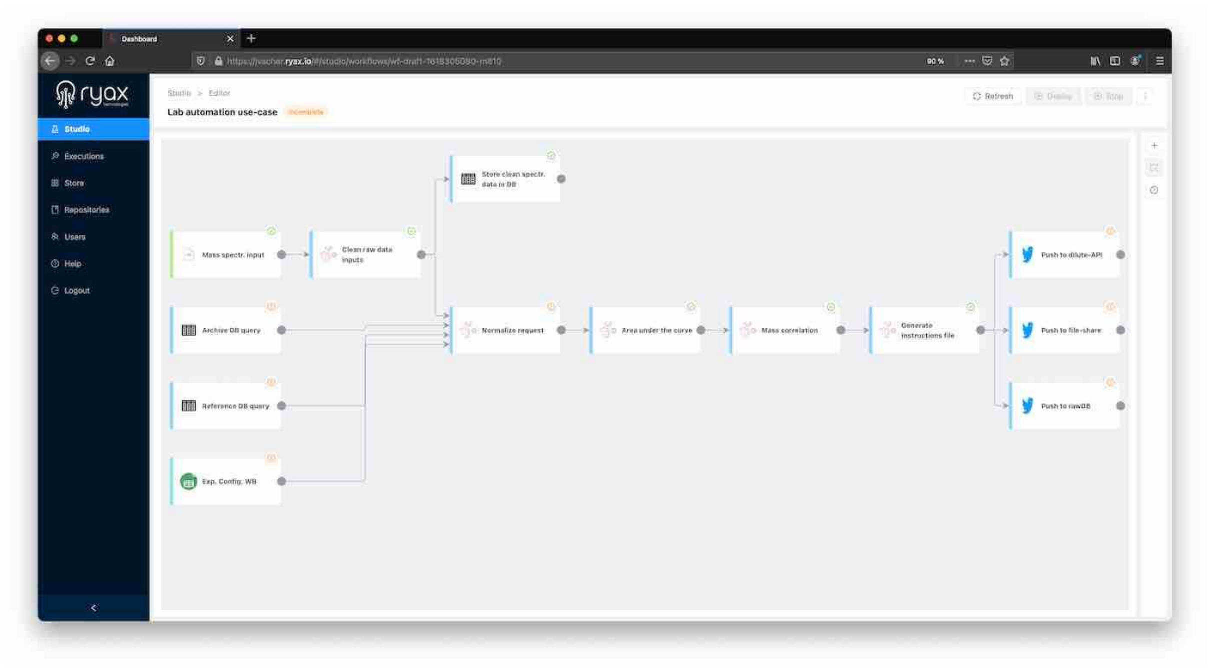


Figure 11. RYAX GUI showing the workflow editor

Each workflow implementation is internally composed by unitary modules considered as a minimal building block. Each of these modules is a stateless software portion that receives data as input and produces data as output.

The modules can be differentiated depending on the role they acquire by the following:

- Sources: code that ingest data from external resources
- Processors: code that execute operations processing data in the workflow
- Publishers: code that push data outside the workflow
- Stream operators: code that perform complex operations in-between modules

Generally, for the code needed on a module implementation, these files are mandatory:

- A metadata file containing information for the module definition on the RYAX tool
- A Python 3 code file containing the logic definition of the module block
- Optionally, if the module needs external dependencies, a requirements.txt could be added

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

The metadata file contains a high-level description of the module and the inputs/outputs using the YAML language.

```yaml
apiVersion: "ryax.tech/v1"
kind: Functions
spec:
  id: tfdetection
  human_name: Tag objects
  detail: "Tag detected objects on images using Tensorflow"
  type: python3
  version: "1.0"
  logo: mylogo.png
  inputs:
  - help: Model name
    human_name: Model name
    name: model
    type: enum
    enum_values:
     - ssdlite_mobilenet_v2_coco_2018_05_09
     - mask_rcnn_inception_v2_coco_2018_01_28
  - help: An image to be tagged; in any format accepted by OpenCV
    human_name: Image
    name: image
    type: file
  outputs:
  - help: Path of the tagged image
    human_name: Tagged image
    name: tagged_image
    type: file
```

Figure 12. Module metadata YAML file snippet example for step (6) Asking for predictions on the AI workflow.

The Python 3 code file contains the module's code including the functions, definitions, and operations that the module needs to perform. In HEROES the logic will be embedded within the Singularity [15] container attached to each step of the workflow (see Section 4.2) the Python code will only serve as a wrapper to launch them. Figure 13 shows an example of what such wrapper could be to submit a Singularity container-based job to the Slurm [19] job scheduler.

```python
import requests
import json
import os
import spython


def handle(req):
    slurm_jwt_token = req["jwt_token"]
    slurm_user = "slurmtest"
    slurm_api = req["slurm_api"]

    headers = {
        "X-SLURM-USER-NAME" : slurm_user,
```

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

```
        "X-SLURM-USER-TOKEN": slurm_jwt_token,
        "Content-Type": "application/json",
    }

    batch_job = """#!/bin/bash
#SBATCH -J singularity_test
#SBATCH -o singularity_test.out
#SBATCH -e singularity_test.err
#SBATCH -p shared
#SBATCH -t 0-00:30
#SBATCH --mem=4000
# Singularity command line options
singularity exec /home/slurmtest/singularity/hello-world.sif cat /etc/os-
release
"""

    # Submit a Slurm Job
    data =  {
        "job": {
            "tasks": "2",
            "name": "test",
            "nodes": "2",
            "current_working_directory": "/tmp/",
            "environment":{
                "PATH":
"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games:/snap/bin",
                "LD_LIBRARY_PATH": "/lib/:/lib64/:/usr/local/lib",
                "HOME": "/home/slurmtest"
            }
        },
        "script": batch_job+curl_call
    }

    resp = requests.post(slurm_api+'/job/submit', headers=headers,
data=json.dumps(data))
    print(resp.text)
    return {}

if __name__ == "__main__":
    handle({
        "slurm_api": "http://slurm-instance.test:9997/slurm/v0.0.36/”,
        "ryax_callback": "https://ryax-instance.test/functions/func-deploy-
1609867549-2xpz/send”
        "jwt_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOxE2MDk4ODU4ODksImlhdCI6MTYwO
Tg2Nzg4OSwic3VuIjoib2N0b3B1cyJ9.NYp7Zu0cahhoL1SogzqYlwIFXRvg0RQVR5Oj7cHeftE
",
        "fileid": "dynamic_output_1608214218675",
    })
```

Figure 13. Module Python code example submitting a Singularity container to a job scheduler

The optional requirements.txt file contains the Python libraries and dependencies required by the module (see example Figure 14).

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

```
requests
json
os
spython
```

Figure 14. Module requirements.txt example

Using the modules as unitary building blocks, a workflow is defined in a textual format also to be used later by the module API in the RYAX tool, obtaining in this format the representation and definition of a complete workflow.

The structure defined on textual format for a workflow is written in a YAML file. This file contains all the workflow definition, differentiating for example: the modules used, the data inputs required by the workflow and other parameters needed for the workflow execution.

```
apiVersion: "ryax.tech/v1"
kind: Workflows
spec:
  human_name: Detect objects on videos
  description: >
      Split video into multiple images, apply tensor flow object
recognition
      and recontruct the video with visual tags of detected objects.
  functions:
    - id: get-video
      from: postgw
      version: "1.0"
      require_external_web_access: true
      position:
        x: 0
        y: 0
      inputs_values:
        introduction: "Submit a video to detect object in it."
      outputs:
        - help: "."
          human_name: "video"
          name: video
          type: file
      streams_to: ["cut-video"]
    - id: cut-video
      from: videoframebyframe
      version: "1.0"
      position:
        x: 1
        y: 1
      inputs_values:
        video: "=get-video.video"
      streams_to: ["detection"]
```

Figure 15. Example YAML file with a workflow definition

RYAX can read the already-defined file and execute it in the correct order as per the internal definition of the different modules, synchronizing their data streams between them and executing the specific computing operations defined to complete the full workflow.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

The definition of modules and workflows using plain text or YAML files, allows and simplifies their version control, management, and code reusability because it can be tracked using a distributed control system like Git.

In the definition and execution of the workflows chosen in the previous sections to test the HEROES platform (see Section 2.3 and Section 3.4), the Python code inside some workflow modules should only contain specific commands to deploy the computational parts inside Singularity containers, to benefit the infrastructure from their benefits, as is described in the next Section 4.2, containerization.

4.2 *Workflows Containerization*

Containerization is defined as a form of operating system virtualization through which applications are run in isolated user spaces called containers, all using the same shared operating system (OS).

A container is essentially a fully packaged and portable computing environment.

The HEROES platform uses containerization on all the steps of the workflows during the execution phase. These containers are managed by the Application and Container Management & Orchestration module, described in details in Deliverable 3.1.

The use of containers enables the immutability of the execution environment and the portability of the software, independently of the destination of the execution. The technology chosen to achieve this is Singularity [15], a well proven solution inside HPC and Cloud environments.

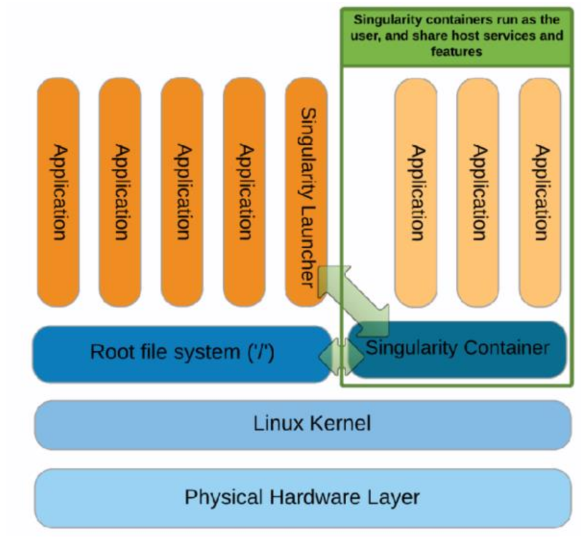


Figure 16. Singularity Architecture

Singularity containers can be built to include all the programs, libraries, data and scripts such that an entire application can be contained and either archived or distributed for others to replicate their behaviour.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

Once a Singularity image container is built by a user, it can be shared over the entire HEROES platform using the Data Management module, that stores the created container images and their information.

This feature allows code reusability and collaborative development, allowing the access and use of a vast catalogue of pre-built container images by the HEROES users, simplifying the process on deploy applications.

More details about the containerization of the applications will be described on the Deliverable 2.2 Workflow Containers, where all the applications needed to execute the containerized workflows will be explained more precisely.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.

5 Conclusion

This document presented the workflows that the HEROES project will use as examples throughout the project. Two types of workflows are represented: one in the HPC field (airflow simulation through CFD simulations), and one in the AI field (image recognition through machine learning).

The steps of each workflow and their integration to the HEROES platform have been described in detail, allowing an introductory design that could be used on the integration phase on the future.

The integration of the workflows, within the "*Workflow and Job management*" module on the HEROES platform, presents a very straight-forward mechanism to build, orchestrate, manage, and deploy functional HPC and ML workflows, simplifying the process for the end-users. The use of Singularity containers makes the execution of each step "independent" of the target platform. The whole workflow execution will be handled by the Ryax software, embedded in the "*Workflow and Job management*" module.

The goal of HEROES is to enable future developments and technology democratization, by providing user-friendly and flexible workflow tools. The main advantages of the workflow execution using the HEROES platform we envision are:

- Savings in working hours due to automatization and customization.
- Allow easy parameterized HPC simulations execution, allowing exploration of new models, parameters…
- The adoption of a workflow manager simplifies the creation, management, and reutilization of pre-created workflows both for HPC and ML.
- The creation of workflows on individual and atomic steps, allows their reutilization and generalization across the entire architecture.
- Simplification of usage for the end-users by providing "pre-packaged" workflows. The usage of a workflow manager and containerization make the whole process of simulation easier, lowering the learning curve and even hiding the execution complexity.
- The containerization of applications allows to share and distribute complete images against other users on the platform, reducing installation and configuration times.

The HEROES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956874.