eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

D5.5 Pillar II - Validation and Evaluation

Version 1.0

Documentation Information

Contract Number	9555558
Project Website	www.eFlows4HPC.eu
Contractual Deadline	29.02.2024
Dissemination Level	[PU]
Nature	Other
Author	Suvarchal K. Cheedela and Nikolay Koldunov
Contributors	Alessandro D'Anca (CMCC), Sonia Scardigno (CMCC), Donatello Elia (CMCC), Rohan Ahmed (BSC), Bruno de Paula Kinoshita (BSC), Suvarchal K. Cheedela (AWI), Nikolay Koldunov (AWI)
Reviewer	Daniele Peano (CMCC)
Keywords	Earth System Models, workflows, data analysis, machine learning

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

Change Log

Version	Description Change
V0.1	Initial ToC
V0.2	First version, sent to internal review
V0.3	Version revised after the internal review
V1.0	Final version, formatted for submission

Table of Contents

1 Executive Summary

This report presents a detailed evaluation of the eFlows4HPC project's Pillar II, focusing on the development and validation of advanced workflows for Earth System Models (ESMs) in High-Performance Computing (HPC) environments. It covers the implementation of dynamic ESM simulation and analysis workflows, incorporating innovative techniques like in-memory data handling, ensemble pruning, and high-performance data analytics. The validation process underscores the workflows' adaptability, fault tolerance, and efficiency in handling large-scale, data-intensive tasks. Through comprehensive assessments, including external evaluations, the report highlights the effectiveness of these workflows in practical ESM applications and acknowledges the challenges faced, particularly in integrating machine learning and managing vast data volumes in HPC settings. The findings suggest significant enhancements in ESM simulations, showcasing the potential of these workflows in advancing climate modeling and analysis, while also pointing towards future avenues for improvement in technology integration and optimization.

2 Introduction

High-performance computing (HPC) has been an integral tool for advancing our understanding of atmospheric and oceanic processes. Over time, it has expanded to encompass Earth System Models, enabling comprehensive simulations that capture the processes within our planet's systems. These simulations stand out as one of the most demanding applications of HPC, not solely because of the immense computational resources they require. They also pose unique challenges, such as managing intensive input/output operations, handling vast volumes of data, and the necessity to process and analyze the data within the same computational environments where it is generated. The advanced workflows of Pillar II have addressed key challenges in ESM simulations on HPC, enhancing data handling, I/O efficiency, and on-site data analysis, potentially leading to more robust and insightful simulations.

2.1 Purpose and Scope of the Report

Pillar II addresses several critical steps in the ESM workflow, namely dynamic data analysis during the model run, and feature extraction during the post-processing phase. In this deliverable, we will describe the fundamental components of the resulting workflows, perform their validation, and evaluate them against the criteria established at the beginning of the project (Deliverable 5.1).

2.2 Outline

We begin by establishing the Workflow Requirements and Validation Methodology, presenting an overview of the initial requirements and the metrics for assessment in Section 3. The core of the report lies in the validation sections, with Section 4 delving into the Dynamic ESM Simulation Workflow Validation, including specific use case applications and the validation process itself, as well as external evaluation. Section 5 mirrors this structure for Analysis and Feature Extraction Validation. The document ends with conclusions and recommendations in Section 6.

3 Workflow Requirements and Validation Methodology

At the start of the eFlows4HPC project, we analyzed standard ESM workflow steps and identified general functional and non-functional requirements and metrics to evaluate the quality of the resulting Pillar II workflows (Deliverable 5.1). This section details the initial set of these Requirements and Metrics, which will inform the evaluation in the subsequent sections.

The framework presented here includes criteria selected for their relevance to the workflows' performance and reliability. The following sections will apply this framework to assess whether the workflows meet the project's objectives, focusing on their practical application within the ESM domain.

3.1 Overview of Initial Requirements and Metrics

Taking into account information on building blocks and requirements of the ESM workflow and its components described above, in this section we provide the general functional and non-functional requirements. The keywords in the priority column are defined according to the RFC 2119 [1].

Table 1. Requirements of the ESM workflow and its components.

ID	Name	Description	Priority
1	Execution Robustness	Management of fault tolerance during the workflow execution including checkpoints or retries. For example, during a large execution if a node fails, the workflow must be able to recover and continue to the end.	Should
2	Portability	Workflow components should be portable to various types of HPC infrastructures.	Should
3	Integrated workflow management	Requires the Management of task dependencies, execution of parallel simulations on different HPC infrastructures, management of batch jobs (submission, monitoring, cancellation), management of conditional paths in a transparent way.	Must
4	Integration with long-term archive/repository storage	Results may be stored in long-term storage for archiving purposes, second use (e.g. downstream services) and/or to satisfy FAIRness policies.	May
5	Workflow adaptability	Capability to easily manage, cancel, replace and add components invocations in the workflow, for instance allowing the execution starting from the n-th step.	Should
6	Access to intermediate in-memory results	The workflow should be able to retrieve data/intermediate outputs of the running processes directly from memory.	Must
7	AI integration for ensemble	Support for applying Machine Learning techniques on intermediate data of running members to compute the members	Should

	member pruning	that will be discarded at a given step of the simulation.	
8	ML/DL capabilities	Requires the support for training and inference of Neural Network models for example for Tropical Cyclone detection.	Must
9	DA capabilities	Support for descriptive analytics (e.g., statistical analysis) exploiting fast in-memory analysis.	Must
10	High Performance Computing support	Climate models have to be executed on computing infrastructures capable of providing a large amount of processing and memory resources.	Must
11	Multi-member analysis	Support for concurrent execution of sub-workflows starting from different inputs (configurable) and comparison of the sub-workflows results.	Must

In line with the requirements, best practices in code development and quality are essential. **Testing**, and in particular continuous integration testing, should be part of the development cycle to catch errors in the workflow. **Inline documentation** aids workflow developers in understanding and adjusting the code. Clear **deployment guides and user manuals** are important for easy installation on new HPC systems and for user adoption of the workflows.

Building on the foundation of these requirements and best practices, we also perform quantitative evaluation of the workflows' success. The subsequent evaluation will employ a defined set of metrics originating from the initial phase of the eFlows4HPC project. Selected for their relevance to the development and operation of workflows, these metrics will quantify aspects such as maintainability, usability, and efficiency. They serve as benchmarks for the assessment, reflecting the project's targets in terms of performance, scalability, and resource utilization. The following table offers a synopsis of these metrics, each with a specific role in the evaluation process, providing a framework for a detailed analysis of the workflows.

Table 2. Metrics for evaluation of ESM workflows.

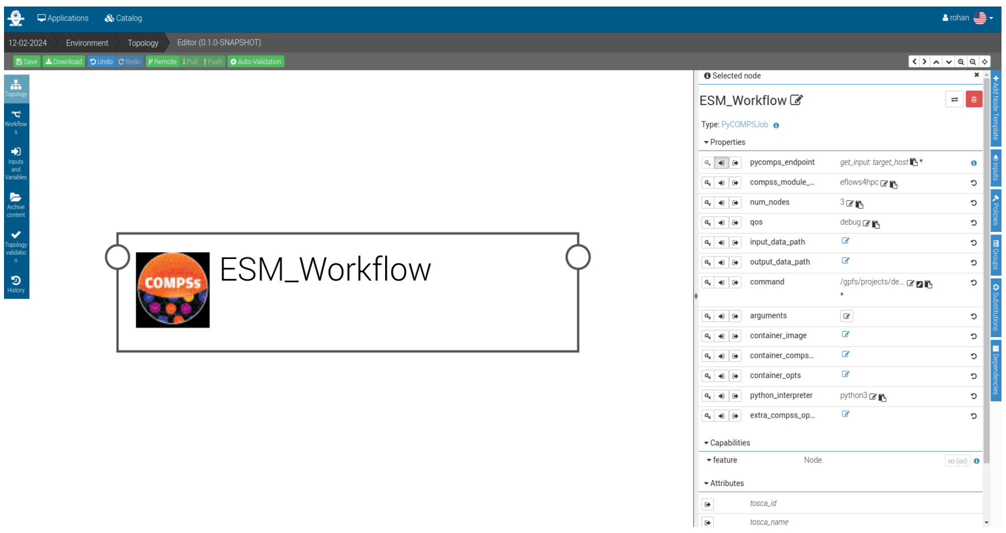
Acronym	Name	Description	Area
LoC	Lines of Code	Number of Lines of code in the workflow implementation.	Development & Maintenance
DoP	Degree of Portability	Percentage of workflow components that can be reused in other infrastructures and workflows.	Accessibility & Deployment
DT	Deployment Time	Time elapsed to deploy the workflow.	Accessibility & Deployment
ET	Execution Time	Time elapsed to execute a workflow.	Performance
SU	Speed-up	Execution time improvement when running with larger resources.	Performance

Eff	Efficiency	Execution time degradation when running larger problems.	Performance
IOT	I/O Time	Percentage of Execution time performing I/O operations.	Data Management
FTC	Fault-tolerant components	Percentage of workflow components that are fault-tolerant.	Reliability
CH	Core/Hour	Number hours of a CPU Core consumed by the workflow execution.	Energy & Cost
EC	Energy Consumption	Energy consumed (Wh or Joules) associated with a workflow execution.	Energy & Cost
AR	Accuracy of the results	Accuracy of scientific results should not degrade.	Pillar II specific
SYPD	Simulated years per day	Throughput of ESM simulations.	Pillar II specific

4 Dynamic ESM simulation workflow Validation

4.1 Workflow Use Case Applications

The dynamic ESM simulation workflow integrates dynamic access to model results and online-based approaches to prune ensemble simulation members during the model execution, with the aim of saving resources. Unlike other standard ESM workflows which do not take into account such on-the-fly decisions based on the generated simulation data from multiple members, this dynamic workflow supports such features thereby proving to be efficient in resource utilization. Moreover, It is available in Alien4Cloud, the web entry point for the workflow, which helps run the workflow with specified parameters over the web interface and abstracting the complexity of workflow initialization at the same time.



Selected node

ESM_Workflow

Type: PyCOMPSJob

Properties

pycomps_endpoint	get_input: target_host *
compss_module_...	eflows4hpc
num_nodes	3
qos	debug
input_data_path	
output_data_path	
command	/gpfs/projects/de... *
arguments	
container_image	
container_comps...	
container_opts	
python_interpreter	python3
extra_compss_op...	

Capabilities

feature Node ∞ (∞)

Attributes

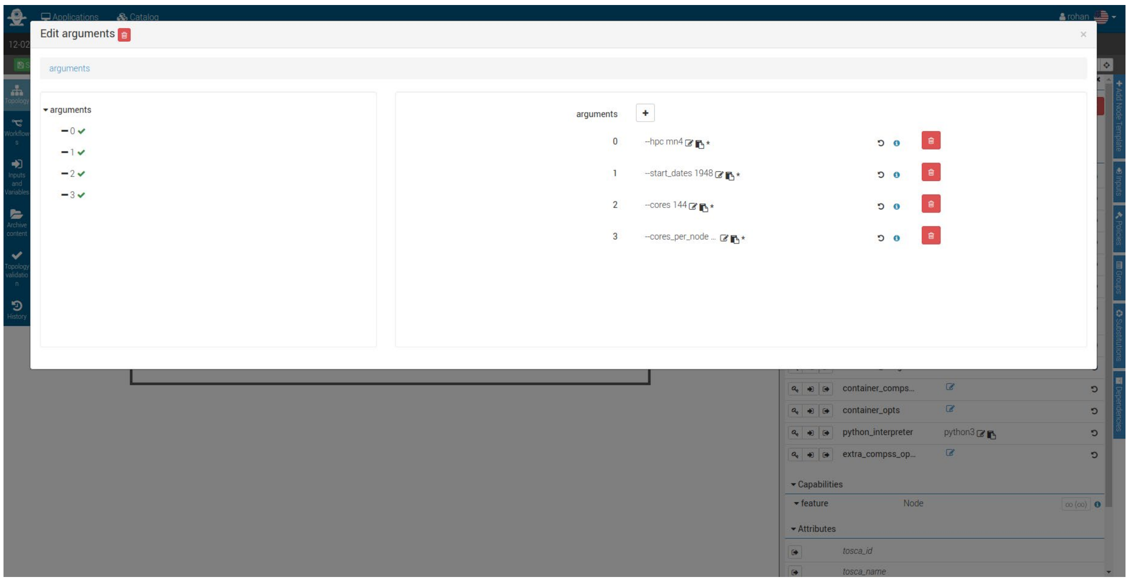
tosca_id

tosca_name

Figure 1. Alien4Cloud screenshot of the ESM_Workflow topology.

Figure 1 shows the TOSCA workflow topology deployed to the Alien4cloud web application. It launches the main workflow Bash Shell script which subsequently loads the required dependencies and modules. Lastly, it calls a PyCOMPSs command (enqueue_compss), ultimately running the Python code for ESM simulation on an HPC platform.

The workflow topology allows the user to pass parameters that define the simulation configuration. Parameters available to the eFlows4HPC user are the HPC platform, list of start dates, number of ensemble members, number of cores to be allocated, and number of cores per node. The screenshot in Figure 2 demonstrates a sample run configuration on the MN4 as the HPC machine, with the start date 1948, 144 as number of cores, and 48 cores per node as computing resources—which results in 3 nodes being allocated for this simulation submission (144 divided by 48).



arguments

- arguments
  - 0
  - 1
  - 2
  - 3

arguments	
0	--hpc mn4
1	--start_dates 1948
2	--cores 144
3	--cores_per_node ...

container_comps...	
container_opts	
python_interpreter	python3
extra_compss_op...	

Capabilities

feature	Node

Attributes

tosca_id

tosca_name

Figure 2. Simulation run configuration, with user provided parameters to control the model configuration.

An intermediate step is handled by scripts written in Bash Shell and Python. These scripts are called by Alien4Cloud and are responsible for validating the user input, loading the required configuration, and preprocessing the model Fortran namelists (which hold configuration values that control input and features of the model). The scripts and the PyCOMPSs workflow definition are both versioned under Git, and available in the eFlows4HPC workflow registry[1].

In Figure 3 we show all the parts of the dynamic ESM simulation workflow. A user request to start the simulation in Alien4Cloud (1) launches the Bash Shell script remotely in MN4 via SSH. The Bash Shell script validates user input, and calls enqueue_compss, a PyCOMPSs utility that starts the workflow (2). One of the tasks in the workflow preprocesses Fortran namelists, replacing variables and creating the random values for the ensemble members (random perturbation) and saves it to the disk for the model (3). Another task is responsible for launching FESOM2 using MPI via the srun Slurm utility (4). The workflow launches one FESOM2 task for each ensemble member.

[1] https://github.com/eflows4hpc/workflow-registry

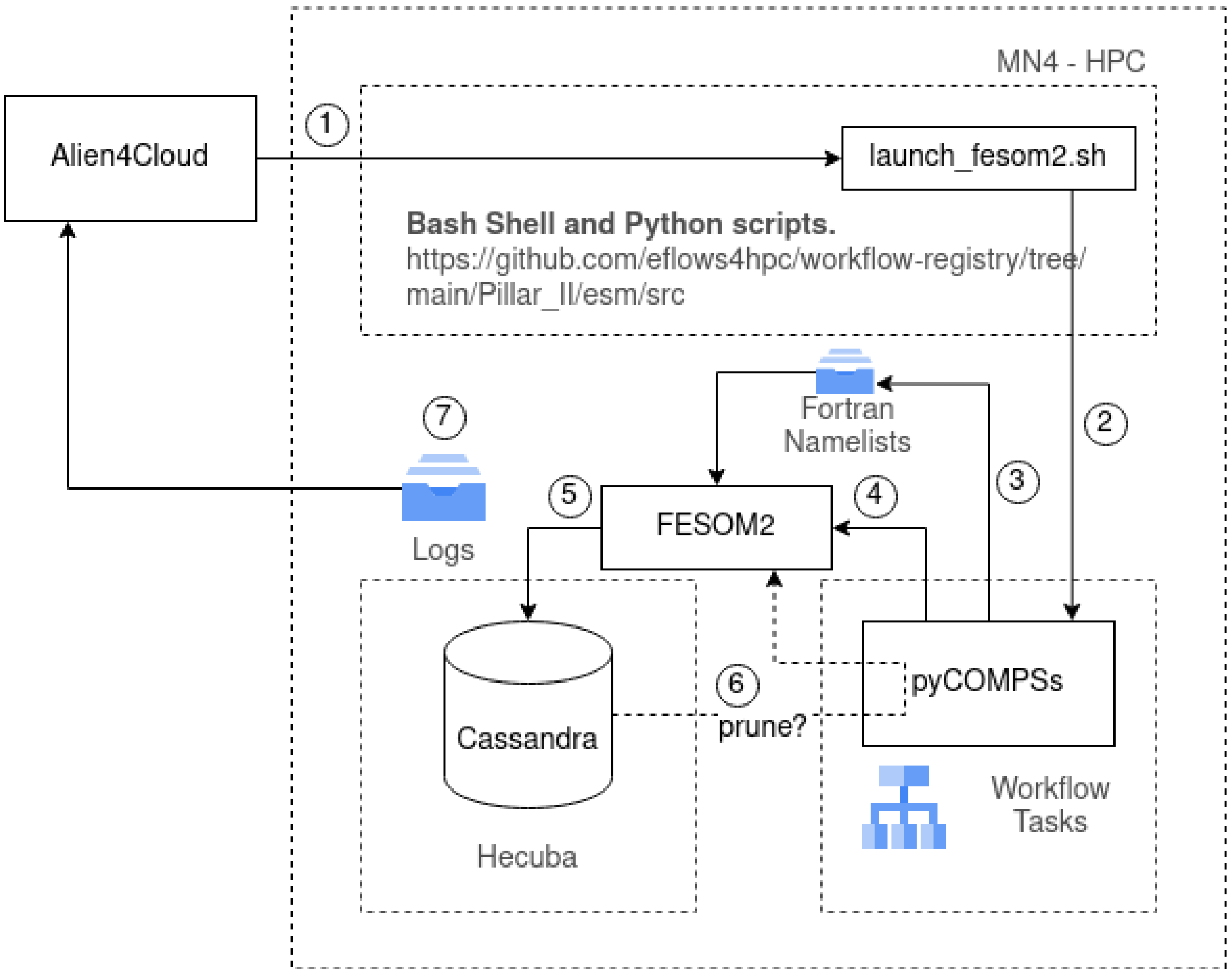
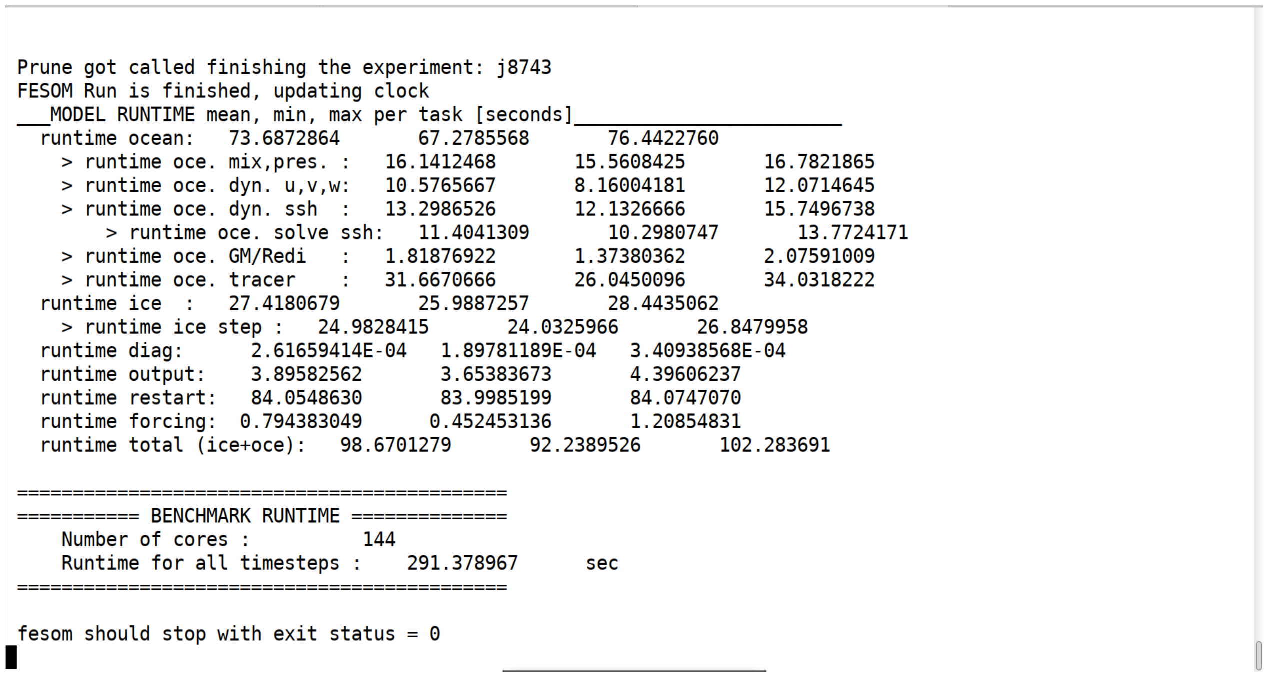


Figure 3. Illustration of all the parts of the dynamic ESM simulation workflow.

FESOM2 reads the preprocessed Fortran namelists that contain paths, and parameters provided by the user that determines the data that is used to start the simulation (input, initial conditions, meshes, simulation parameters and output variables). FESOM2 writes requested output variables at requested frequency to the Cassandra database using the Hecuba API (5). The data written by FESOM is retrieved by another PyCOMPSs task responsible to evaluate whether ensemble members must be pruned or not (6). This task is responsible for ML/AI component used to back propagate the changes in timestep when simulation is unstable. Most parts of the execution create logs on the disk in MN4, which are available to the user during the simulation via the Alien4Cloud web interface (7), and after the simulation as they are stored in the HPC platform on a path configured by the user.

The simulations executed on MN4 used the FESOM2 “pi” test mesh, and the “core2” mesh (meshes are described in the deliverable “D9.2 Data Management Plan version 3.0”, under “Data Summary”). The resources used for these simulations were, respectively, 1 node with 2 cores, and 3 nodes with 144 cores. FESOM2 does not output NetCDF, as, instead, it writes to an instance of a Cassandra database managed by Hecuba and pyCOMPSs (as illustrated in Figure 3). It is important to note that the model is not compiled during the workflow execution. Instead, the workflow uses a pre-compiled version of FESOM2 on MN4, from the FESOM2 public repository Git branch eflows_hecuba_templates_update[2].

For pruning, a pyCOMPSs task is initialized with the rest of the workflow, and is responsible to call the code that analyzes the status of ensemble members and mark them to be pruned or not. To mark an ensemble member to be pruned, the task changes a flag in the Cassandra database. Doing that, the next time the FESOM2 model queries the database, it also checks for the flag and if it corresponds to the ensemble member being pruned the model stops, as seen in Figure 4.



```
Prune got called finishing the experiment: j8743
FESOM Run is finished, updating clock
___MODEL RUNTIME mean, min, max per task [seconds]________________________
  runtime ocean:   73.6872864       67.2785568       76.4422760
    > runtime oce. mix,pres. :   16.1412468       15.5608425       16.7821865
    > runtime oce. dyn. u,v,w:   10.5765667       8.16004181       12.0714645
    > runtime oce. dyn. ssh  :   13.2986526       12.1326666       15.7496738
        > runtime oce. solve ssh:   11.4041309       10.2980747       13.7724171
    > runtime oce. GM/Redi   :   1.81876922       1.37380362       2.07591009
    > runtime oce. tracer    :   31.6670666       26.0450096       34.0318222
  runtime ice  :   27.4180679       25.9887257       28.4435062
    > runtime ice step :   24.9828415       24.0325966       26.8479958
  runtime diag:      2.61659414E-04   1.89781189E-04   3.40938568E-04
  runtime output:    3.89582562       3.65383673       4.39606237
  runtime restart:   84.0548630       83.9985199       84.0747070
  runtime forcing:  0.794383049      0.452453136       1.20854831
  runtime total (ice+oce):   98.6701279       92.2389526       102.283691

============================================
=========== BENCHMARK RUNTIME ==============
    Number of cores :          144
    Runtime for all timesteps :    291.378967      sec
============================================

fesom should stop with exit status = 0
```

Figure 4. Pruning an ensemble member task called from the terminal.

4.2 Requirements validation

The table of this section is a copy of Table 1 from section 3.1 “Overview of Initial Requirements and Metrics”, with the additional column “Status”. This new column contains the validation status for each requirement in the context of the Dynamic ESM simulation workflow.

Table 3. Dynamic ESM simulation workflow requirements validation

ID	Name	Description	Priority	Status
1	Execution Robustness	Management of fault tolerance during the workflow execution including checkpoints or	Should	Implemented via pyCOMPSs fault-tolerance feature[3] and FESOM2’s ability to checkpoint and restart the simulations.

[2] https://github.com/FESOM/fesom2/tree/eflows_hecuba_templates_update

[3] https://pycompss.readthedocs.io/en/stable/Sections/09_PyCOMPSs_Notebooks/syntax/3.4_Defining_classes_and_objects-with-fault-tolerance.html

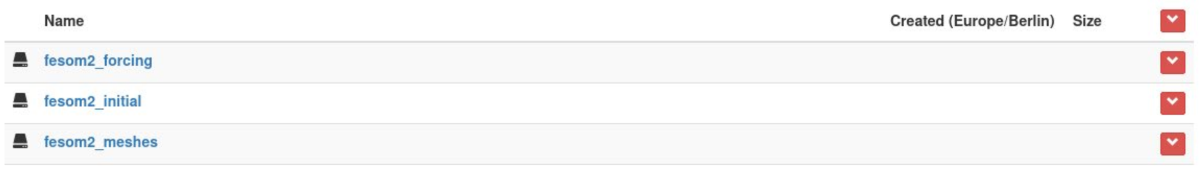
		retries. For example, during a large execution if a node fails, the workflow should be able to recover and continue to the end.		
2	Portability	Workflow components should be portable to various types of HPC infrastructures.	Should	Implemented with Singularity containers deployed in MN4, Spack specification for installing all necessary dependencies on any HPC and modular configuration implemented in Bash Shell and Python scripts[4].
3	Integrated workflow management	Requires the Management of task dependencies, execution of parallel simulations on different HPC infrastructures, management of batch jobs (submission, monitoring, cancellation), management of conditional paths in a transparent way.	Must	Implemented with pyCOMPSs, and parametrization of Alien4Cloud application and of Bash Shell and Python scripts.
4	Integration with long-term archive/repository storage	Results may be stored in long-term storage for archiving purposes, second use (e.g. downstream services) and/or to satisfy FAIRness policies.	May	Through the use of Hecuba and a Cassandra database, results can be stored in a large persistent database, or transferred to other locations such as cloud storage.
5	Workflow adaptability	Capability to easily manage, cancel, replace and add components invocations in the workflow, for instance allowing the execution starting from the n-th step.	Should	Implemented through parametrization of PyCOMPSs workflow, where resources, model configuration and analysis can be changed.
6	Access to intermediate in-memory results	The workflow must be able to retrieve data/intermediate outputs of the running processes directly from	Must	Access to in-memory data is provided with a numpy-like array view using HECUBA, and is further abstracted using Dask[9] arrays and xarray[7] for efficient access.

[4] https://github.com/eflows4hpc/workflow-registry/blob/main/Pillar_II/esm/README.md

		memory.		
7	AI integration for ensemble member pruning	Support for applying Machine Learning techniques on intermediate data of running members to compute the members that will be discarded at a given step of the simulation.	Should	AI technique is used to adaptively change the time step for each ensemble member . Pruning is done based on the stability of each ensemble member.
8	ML/DL capabilities	Requires the support for training and inference of Neural Network models for example for Tropical Cyclone detection.	Must	Choice of representing the in-memory data numpy-like format makes it amenable to many AI/ML methods and tools.
9	DA capabilities	Support for descriptive analytics (e.g., statistical analysis) exploiting fast in-memory analysis.	Must	Choice of representing the in-memory data numpy-like format makes it amenable to extensive statistical functions of Numpy and SciPy. Additionally, use of Dask[9] arrays paves way for an efficient and scalable analysis.
10	High Performance Computing support	Climate models have to be executed on computing infrastructures capable of providing a large amount of processing and memory resources.	Must	Climate model is containerized and has environments defined for numerous HPC environments. PyCOMPSs workflows are tested on MN4 and on Levante with Slurm.
11	Multi-member analysis	Support for concurrent execution of sub-workflows starting from different inputs (configurable) and comparison of the sub-workflows results.	Must	With the parametrization that was pending in the previous deliverable we are now able to have multiple users starting simulations concurrently, re-using the same input data and model binaries and containers.

The dynamic ESM simulation workflow code has been written with an IDE that provides a free code inspection tool (JetBrains PyCharm), and other non-functional requirements such as coding best practices have been followed. For example, the project uses ShellCheck and bats for static analysis and unit tests for Bash Shell scripts. And it uses Mypy for type checking and Pytest for unit tests in Python. The current code coverage is at 76.59%. These tools are executed in continuous integration for every commit to the workflow using GitHub Actions. Finally, the workflow folder in GitHub includes a README.md with documentation with general information about the workflow, instructions to build, run, and troubleshoot the workflow.

- **Execution robustness:** The use of PyCOMPSs provides fault tolerance by capturing error codes from failed tasks and facilitating their re-execution. Similarly, the FESOM2 model enables the periodic saving of the model's state, called restarts, at user-defined intervals in its namelist. This feature allows for the resumption of execution from the last saved state. In the event of a failure, the PyCOMPSs workflow initiates task retries, and the FESOM2 model resumes the simulation from the last saved checkpoint, thereby ensuring continuity and eliminating the loss of progress.
- **Portability:** It is achieved by containerizing the workflow, including all applications of the workflow. Containerizing the entire workflow is done using the HPCWaaS service provided by eFlows4HPC. This service employs Spack specifications (https://github.com/eflows4hpc/software-catalog/tree/main/packages) to construct containers in the Singularity format tailored for specific HPC targets. As a result, it ensures compatibility with MPI-runtime environments and facilitates secure deployment on the intended HPC systems. While the A4C framework, in conjunction with Yorc, enables deployment and execution on chosen architectures, it has been primarily tested on the MN4 system. Additionally, the applications have been independently verified and constructed using consistent methodologies on the Levante HPC at the German Climate Computing Center and local linux desktop development environments.
- **Integrated workflow management:** PyCOMPSs is instrumental in initiating and managing essential components such as Cassandra, which is crucial for storing and handling the simulations' in-memory data, simulations and analysis tasks. At the outset of each simulation, PyCOMPSs is responsible for bootstrapping Cassandra, ensuring that the database is up and running to accommodate the data influx from the simulations.
- **Integration with Long-term Archive/Repository Storage:** The dynamic Earth system model (ESM) simulation workflow incorporates a generic and extendable strategy. Specifically, the inputs required for model simulations—categorized broadly into initial conditions, forcing, and mesh—are archived in cloud storage available at swift.dkrz.de.https://swiftbrowser.dkrz.de/public/dkrz_035d8f6ff058403bb42f8302e6badfbc/eflows4hpc/



Name	Created (Europe/Berlin)	Size
fesom2_forcing		
fesom2_initial		
fesom2_meshes		

Figure 5. Input datasets stored on DKRZ's swift storage, these can be downloaded on-the-fly in any workflow to start simulations using FESOM2.

These input datasets can be accessed using a Python function as part of the pre-simulation task within the PyCOMPs workflow. (PyCompss task figure follows).

Similarly, simulation outputs are archived in a designated cloud storage container, systematically organized by simulation ID, time step, and spatial chunk. This approach not only facilitates the parallel uploading of data required for subsequent analysis but also adopts a format akin to the

Zarr data format. This choice enhances the ease of data integration and analysis in Python by a broader scientific community.

```python
@task(filePath=FILE_INOUT)
def download_swift_files(directory, url):
    # Check if the directory exists, create it if not
    if not os.path.exists(directory):
        os.makedirs(directory)
    # Get the HTML content from the URL
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    # Find all links in the page
    for link in soup.find_all('a'):
        # Construct the full URL
        file_url = urljoin(url, link.get('href'))
        # Check if the link is for a .nc file
        if file_url.endswith('.nc'):
            file_name = os.path.basename(file_url)
            file_path = os.path.join(directory, file_name)

            if not os.path.exists(file_path):
                print(f"Downloading {file_name}...")
                with requests.get(file_url, stream=True) as file_response:
                    with open(file_path, 'wb') as file:
                        for chunk in file_response.iter_content(chunk_size=8192):
                            file.write(chunk)
                print(f"Downloaded {file_name}")
            else:
                print(f"{file_name} already exists.")
url_prefix="https://swiftbrowser.dkrz.de/public/dkrz_035d8f6ff058403bb42f8302e6badfbc/eflows4hpc/"

download_swift_files('initial_files', url_prefix+"fesom2_initial/phc3.0/?show_all")
download_swift_files('forcing_files', url_prefix+"fesom2_forcing/?show_all")
download_swift_files('mesh_files',    url_prefix+"fesom2_meshes/core2/?show_all")
```

Figure 6. Illustrates workflow task to download necessary input data required for FESOM2 simulation from a cloud store before simulation.

```python
def upload_to_swift(basename, container_name, variable_name, timeint, chunkint, data):
    auth_token = os.getenv('OS_AUTH_TOKEN')
    storage_url = os.getenv('OS_STORAGE_URL')

    if not auth_token or not storage_url:
        raise ValueError("OS_AUTH_TOKEN and OS_STORAGE_URL must be set in environment variables.")

    # Create the container if it doesn't exist
    create_container_if_needed(storage_url, container_name, auth_token)

    # Construct the object path
    object_path = f"{basename}/{container_name}/{variable_name}/{timeint}/{chunkint}"
    upload_url = f"{storage_url}/{basename}/de" #{object_path}"

    headers = {
        'X-Auth-Token': auth_token,
        'Content-Type': 'application/octet-stream'
    }

    response = requests.put(upload_url, headers=headers, data=data)

    if response.status_code == 201:
        print("Upload successful.")
    else:
        print(f"Upload failed with status code: {response.status_code}")
        print(f"Response: {response.text}")
```

Figure 7. Illustrates workflow task to upload the simulation results to cloud storage either during simulation or at the end of the simulation.

- **Workflow adaptability:** This requirement emphasizes the flexibility to modify the workflow in response to changes in model configurations, such as alterations in resolution, or adjustments in analysis procedures. It also pertains to the workflow's capacity to operate effectively with variations in data volume during input/output (IO) operations. The workflows have been evaluated for their robustness with increase in IO frequency and when applied to model configurations of higher spatial resolution. Moreover, the procurement of required inputs for higher-resolution models is streamlined by the methodologies outlined in the section on Integration with Long-term Archive/Repository Storage, ensuring a cohesive and adaptable simulation environment.

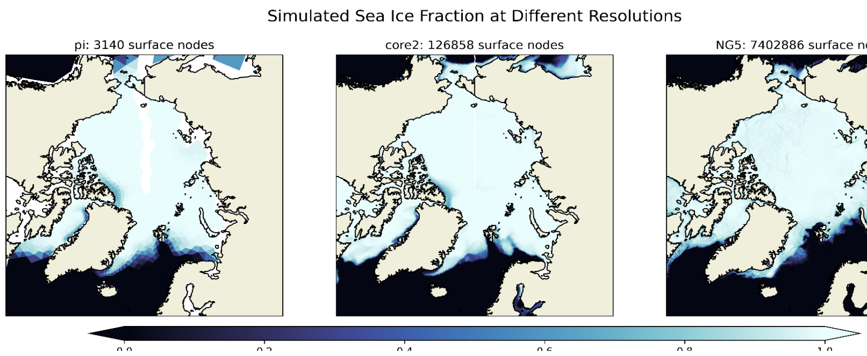
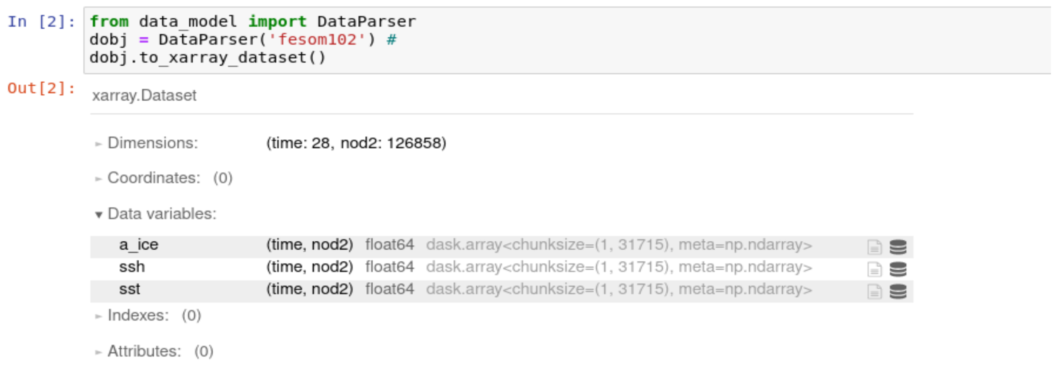

Simulated Sea Ice Fraction at Different Resolutions
pi: 3140 surface nodes
core2: 126858 surface nodes
NG5: 7402886 surface nodes

Figure 8. Simulated sea-ice fraction from test configuration (left, 3140 surface nodes), climate configuration (~120K surface nodes), and high resolution configuration (~7.3M surface nodes). These configurations can easily be changed within the workflow.

- **Access to intermediate in-memory results:** The core of the dynamic Earth System Model (ESM) workflow is the in-memory analysis of data stored within Cassandra. HECUBA provides a cross-language, user-friendly API for Cassandra, facilitating the efficient storage and retrieval of data. Since our previous deliverable, we have improved the IO throughput by adding the ability to ingest data from model simulations into Cassandra in parallel. This data is stored as binary blobs, organized by variable name and time chunks, a method consistent with the Zarr format. This compatibility provides scalable access and analysis of the data in Python through xarray[7], where the array views are represented as lazy-loading Dask arrays.



```
In [2]: from data_model import DataParser
        dobj = DataParser('fesom102') #
        dobj.to_xarray_dataset()

Out[2]: xarray.Dataset

        ▸ Dimensions:     (time: 28, nod2: 126858)
        ▸ Coordinates:  (0)
        ▾ Data variables:
            a_ice        (time, nod2)  float64  dask.array<chunksize=(1, 31715), meta=np.ndarray>
            ssh          (time, nod2)  float64  dask.array<chunksize=(1, 31715), meta=np.ndarray>
            sst          (time, nod2)  float64  dask.array<chunksize=(1, 31715), meta=np.ndarray>
        ▸ Indexes:  (0)
        ▸ Attributes:  (0)
```

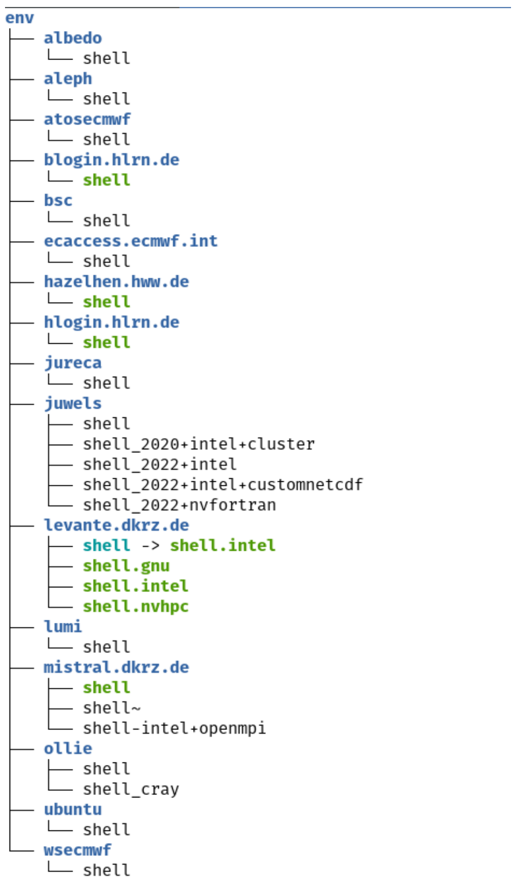
Figure 9. Illustration of in-memory data represented as an Xarray dataset: it shows the conversion of simulation results into an Xarray dataset format, showcasing dimensions, data variables, and their respective underlying dask array structures for lazy, and efficient parallel computation. These arrays are compatible with numerous statistical functions of numpy and scipy.

- **AI integration for ensemble member pruning:** In the context of dynamic ESM workflows, ensemble members are generated by varying initial conditions or altering model parameters. Each member undergoes an in-memory analysis to determine its relevance and accuracy, a process crucial for pruning less informative or redundant simulations to optimize computational resources.

  The analysis phase introduces a novel approach by incorporating AI/ML techniques to refine the ensemble. Specifically, during the simulation phase, an AI/ML-based methodology is employed to adaptively modify the simulation's timestep. This adjustment aims to reduce the spin-up time for ensemble simulations, enhancing efficiency without compromising the quality of results.

  The adaptive timestep approach utilizes JAX, a high-performance machine learning library, to fine-tune a control parameter influencing the timestep size based on the deviation between current simulation outcomes and a set of reference data. This deviation is quantified using metrics such as the Mean Squared Error (MSE) of coarse grained simulation data, providing a clear measure of the simulation's current performance against expected ranges of simulation output.

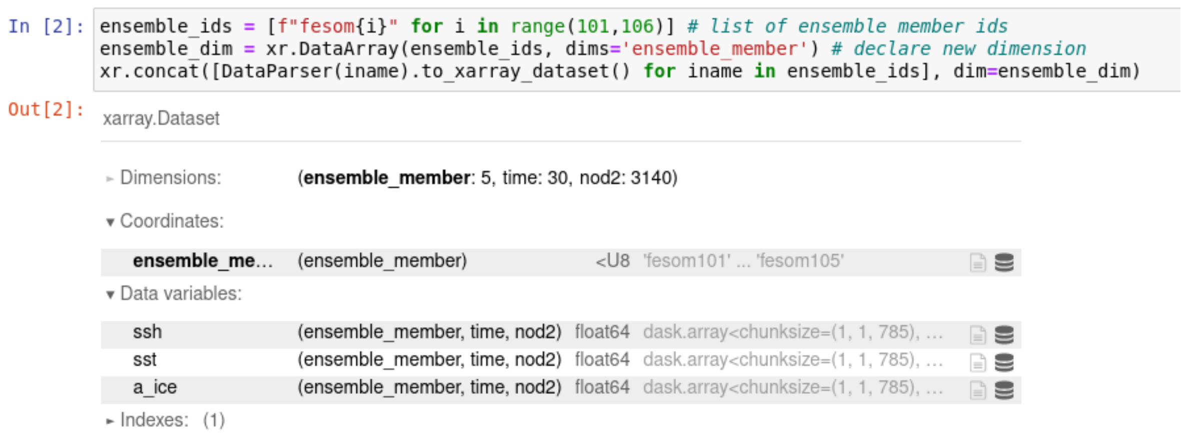
  Through the application of backpropagation technique, the control time step parameter is iteratively refined. By employing gradient descent algorithm, each update aims to reduce the error between the simulation and reference data, thereby optimizing the timestep size. This process ensures that the timestep is continually adjusted in response to the evolving conditions of the simulation, maintaining both stability and accuracy.
- **ML/DL capabilities:** Simple, novel, ML based method is used in the analysis to dynamically change the time step as described in the section AI integration for ensemble member pruning. It also illustrates further ML possibilities that can take advantage of the representation of in-memory data represented in a convenient numpy-like format amenable to many AI/ML methods.
- **DA capabilities:** This requirement aims to incorporate widely-used statistical analysis tools into the in-memory data analysis. By making the simulation data from Cassandra available as an Xarray dataset in Python, as described in above section, Access to intermediate in-memory results, it seamlessly integrates with the extensive statistical functions provided by libraries such as NumPy and SciPy. Furthermore, constructing the dataset as lazy data using Dask arrays enhances efficiency, enabling more effective parallel computation for statistical analysis. This approach significantly streamlines the process, allowing for sophisticated data manipulation and examination directly within the in-memory results.
- **High-Performance Computing Support:** As mentioned in the above section on Portability, the ESM workflow benefits from containerization. These containers can be constructed using a container service tailored for HPC centers, enabling their use across various HPC environments. Additionally, the FESOM2 model, used across numerous HPC centers across Europe, comes with pre-defined environments that are regularly updated. This ensures that the model can be compiled and executed smoothly on these platforms, reinforcing the workflow's adaptability and performance on advanced computing infrastructures. Furthermore, the containerized version of the FESOM2 model is regularly used for the continuous-integration tests in its source code repository. This container is portable across HPC centers with minor changes in the build step.



```
env
├── albedo
│   └── shell
├── aleph
│   └── shell
├── atosecmwf
│   └── shell
├── blogin.hlrn.de
│   └── shell
├── bsc
│   └── shell
├── ecaccess.ecmwf.int
│   └── shell
├── hazelhen.hww.de
│   └── shell
├── hlogin.hlrn.de
│   └── shell
├── jureca
│   └── shell
├── juwels
│   ├── shell
│   ├── shell_2020+intel+cluster
│   ├── shell_2022+intel
│   ├── shell_2022+intel+customnetcdf
│   └── shell_2022+nvfortran
├── levante.dkrz.de
│   ├── shell -> shell.intel
│   ├── shell.gnu
│   ├── shell.intel
│   └── shell.nvhpc
├── lumi
│   └── shell
├── mistral.dkrz.de
│   ├── shell
│   ├── shell~
│   └── shell-intel+openmpi
├── ollie
│   ├── shell
│   └── shell_cray
├── ubuntu
│   └── shell
└── wsecmwf
    └── shell
```

Figure 10. Illustrates HPC support for using FESOM2 model. The env directory included with FESOM2 source code contains configurations for different HPC centers that are regularly updated, at the lowest level is a shell file which contains all machine specific environments for that HPC.

- **Multi-member analysis:** This requirement focuses on incorporating support for multi-member analysis into the in-memory data framework as necessary. By making the results of each simulation member accessible as Xarray datasets, we can utilize Xarray's robust functionality to integrate multiple datasets along a newly defined ensemble dimension. This enables analysis across different ensemble members.



```
In [2]: ensemble_ids = [f"fesom{i}" for i in range(101,106)] # list of ensemble member ids
        ensemble_dim = xr.DataArray(ensemble_ids, dims='ensemble_member') # declare new dimension
        xr.concat([DataParser(iname).to_xarray_dataset() for iname in ensemble_ids], dim=ensemble_dim)

Out[2]: xarray.Dataset

        ▸ Dimensions:      (ensemble_member: 5, time: 30, nod2: 3140)
        ▾ Coordinates:
          ensemble_me...   (ensemble_member)               <U8 'fesom101' ... 'fesom105'
        ▾ Data variables:
          ssh              (ensemble_member, time, nod2)   float64 dask.array<chunksize=(1, 1, 785), ...
          sst              (ensemble_member, time, nod2)   float64 dask.array<chunksize=(1, 1, 785), ...
          a_ice            (ensemble_member, time, nod2)   float64 dask.array<chunksize=(1, 1, 785), ...
        ▸ Indexes: (1)
```

Figure 11. Illustrates use of xarray's features to merge in-memory results from multiple ensemble members to a single dataset for analysis over ensemble dimension.

4.3 Metrics evaluation

The table of this section is a copy of Table 2 from section 3.1 “Overview of Initial Requirements and Metrics”, with the additional column “Value”. This new column contains the value for each metric assessed for the Dynamic ESM simulation workflow.

Table 4. Dynamic ESM simulation workflow metrics evaluation

Acronym	Name	Description	Value
LoC	Lines of Code	Number of Lines of code in the workflow implementation.	2951 total Python ESM code 720 code, 287 comments 635 Python 242 Bash Shell 12 INI Alien4Cloud topology 118 YAML Changes in FESOM2 937 Fortran
DoP	Degree of Portability	Percentage of workflow components that can be reused in other infrastructures and workflows.	100%, with containers and parameterized execution.
DT	Deployment time	Time elapsed to deploy the workflow	Under 2 minutes
ET	Execution Time	Time elapsed to execute a workflow.	160 seconds on average, with 3 nodes and 144 cores on MareNostrum4 (excluding queue time).
SU	Speed-up	Execution time improvement when running with larger resources.	Linear scaling for up to 8000 cores at core2 configuration (120K surface nodes) resolution and up to ~35K cores for NG5 configuration (~7.3M surface nodes). FESOM2 has been well tested for scaling in publications[8].

Eff	Efficiency	Execution time degradation when running larger problems.	100 up to the scaling limit.
IOT	I/O Time	Percentage of Execution time performing I/O operations.	55%
FTC	Fault-tolerant components	Percentage of workflow components that are fault-tolerant.	100% through pyCOMPSs fault-tolerance feature.
CH	Core/Hour	Number hours of a CPU Core consumed by the workflow execution.	0.02
EC	Energy Consumption	Energy consumed (Wh or Joules) associated with a workflow execution.	2532 W
SYPD	Simulated years per day	Throughput of ESM simulations	60 SYPD on 400 cores without IO for low-res climate configuration, core 2 (~120K surface nodes). 6 SYPD for km-scale configuration (~7.3M surface nodes) on 25000 cores.

- **LoC:** Lines of code are calculated with an Open Source tool called pygount[5], version 1.6.1. To replicate these numbers, one can clone the workflow-registry repository, checkout the main branch, git commit 3275bca60b473abe40378527b250609c4cb75d01, and run `pygount --format=summary --folders-to-skip awicm3 --suffix "yaml,yml,py,ini,sh" .`. For the FESOM2 code part, one can clone the repository, switch branches with `git checkout eflows_hecuba_templates_update`, then use `git log --numstat` to retrieve the number of lines added and removed, and apply some simple Bash Shell scripting to subtract removed from added and sum everything to have the number of lines modified: `git log --numstat --pretty="%H" master..eflows_hecuba_templates_update | grep -o -P "\d\s+\d" | awk '{print $1-$2}' | awk '{s+=$1} END {printf "%d\n", s}'`.
- **DoP:** The FESOM2 model was executed on both MN4 and Levante, following the compilation instructions from the project documentation (basically issuing a command like ./configure.sh bsc). The rest of the workflow was partially tested on Levante. Alien4Cloud was used to connect and launch simple jobs in Levante, however, COMPSs and Hecuba were not successfully installed on Levante, impeding an end-to-end test on Levante. These issues are related to the installation of infrastructure and tools for the workflow, but the workflow itself and dependencies are fully portable, also via Singularity containers.
- **DT:** The deployment time for the Dynamic ESM simulation workflow was measured from the time a logged-in user in Alien4Cloud started the process to deploy the workflow application using an existing Alien4Cloud Topology (as described in the workflow registry

[5] https://pypi.org/project/pygount/

documentation, and in previous deliverables). It does not account for the time to deploy Alien4Cloud, FESOM2, HPC modules, Python, and other dependencies and tools used in the workflow.

- **ET:** The value provided is for the execution of the workflow on MN4 with the “core2” mesh, 144 cores, using 3 nodes. The simulation used a single ensemble member for the start date 1948.
- **SU:** Speed-up represents the improvement in execution time when the workflow is executed using more computing resources FESOM2 scales linearly up to 400 cores at core2 configuration (120K surface nodes) and up to 25K cores for km-scale configuration, NG5.
- **Eff**: Efficiency is an index that represents the degradation of computation time when executing larger problems. Up to above mentioned linear scaling limits, efficiency is 100.
- **IOT**: In the context of in-memory analysis, the input/output (IO) process can present a significant overhead when compared to conventional IO methods. This is due to the relatively high-frequency data required for the dynamic analysis step, coupled with the immediate disposal of data post-analysis, which results in considerable storage space savings. To evaluate this, a series of tests were conducted comparing the traditional serial IO backend (netCDF) with the HECUBA backend within the FESOM2 model. The findings revealed that at a relatively low-resolution (core2), the netCDF output occupied approximately 40% of the total simulation time, whereas HECUBA accounted for about 55%. This discrepancy suggests that the parallel IO strategy may be inefficient, potentially due to the transmission of excessively small data chunks to HECUBA, resulting in considerable time lost in establishing connections to Cassandra. Although it has yet to be verified, there is a hypothesis that the current parallel, asynchronous IO approach implemented with HECUBA could yield superior performance in higher resolution simulations compared to the netCDF backend.
- **FTC:** Fault tolerance is fulfilled by utilizing pyCOMPSs alongside the embedded checkpointing feature within the FESOM2 model. PyCOMPSs detects the exit code of the failed task, enabling its re-execution without restarting the entire process; the FESOM2 model includes an internal checkpointing mechanism by saving its state using a restart file. So we can consider the percentage of FTC equal to 100%.
- **CH:** Core/Hour represents the number of hours of a CPU Core consumed by the workflow execution, as retrieved from the BSC HPC Portal for a successful simulation. The BSC HPC Portal stores information saved by Slurm in MareNostrum4.
- **EC:** Energy Consumption represents the energy consumed (Wh or Joules) associated with a workflow execution, as retrieved from the BSC HPC Portal for a successful simulation. The BSC HPC Portal stores information saved by Slurm in MareNostrum4. Below, Figure 12, shows the power consumption (in Watts) comparison of the dynamic ESM workflow and running the same ESM workflow without the pruning mechanism.

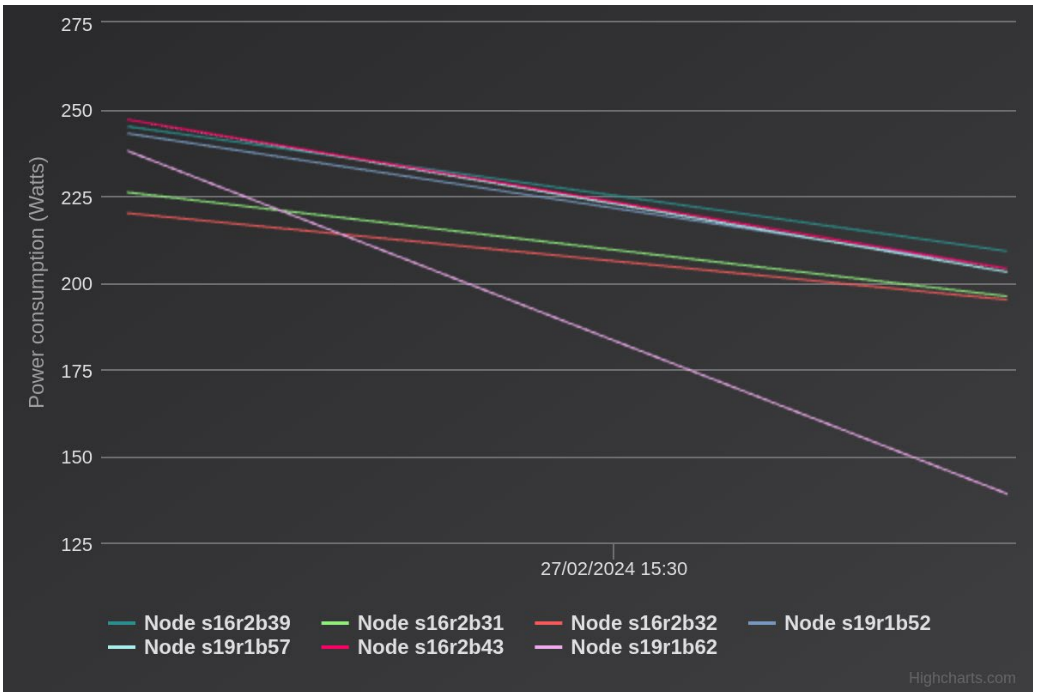


Figure 12. Dynamic ESM workflow power consumption with pruning mechanism enabled on MN4.

It shows power consumption of each node with respect to time by running the dynamic ESM workflow simulation and by pruning a member, the power consumption gets reduced significantly thereby saving energy resources leveraging dynamicity of the ESM workflow.

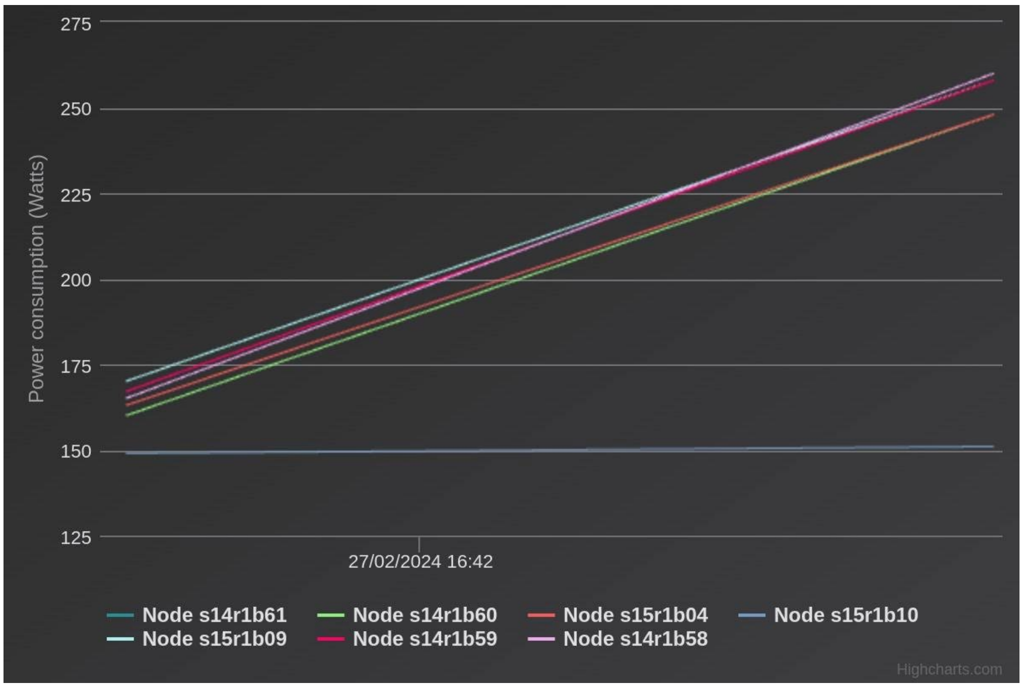


Figure 13. Dynamic ESM workflow (without pruning mechanism enabled) power consumption on MN4.

Figure 13 shows power consumption by running the same workflow but without the pruning mechanism thereby acting as a standard ESM workflow. Since it lacks the dynamicity of pruning a member at the runtime of workflow, saving power resources does not happen in these cases.

Likewise, CPU usage between two workflow runs are also displayed in Figure 14, below.

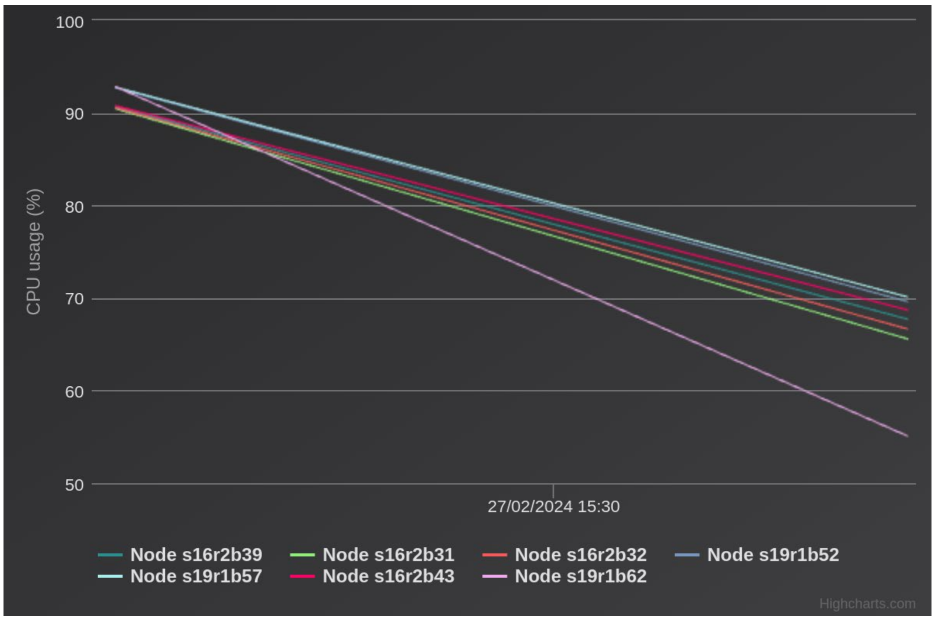


Figure 14. Dynamic ESM workflow CPU usage on MN4.

The overall CPU usage reduced significantly during the dynamic ESM workflow simulation as compared to the run without pruning mechanism in which the CPU usage remains almost static as illustrated in Figure 15 below.

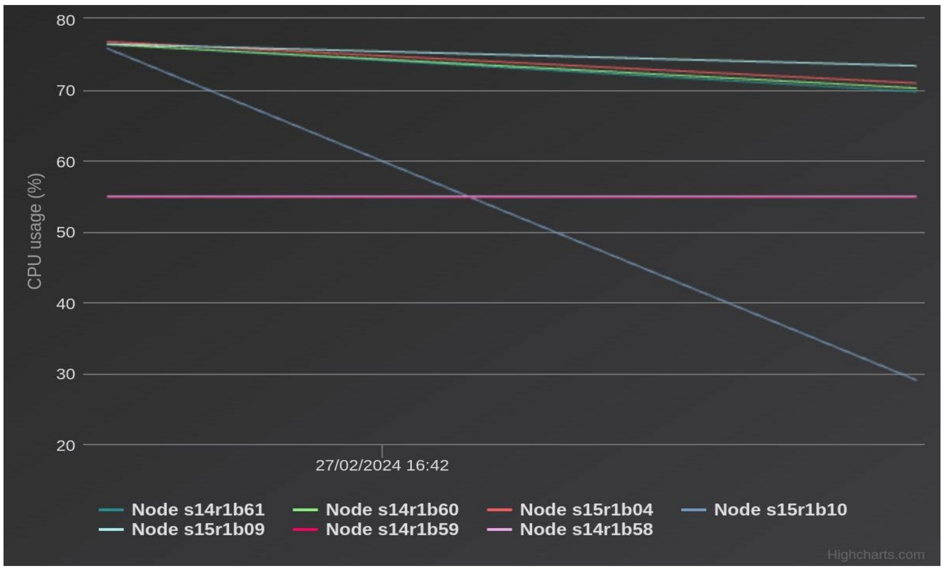


Figure 15. Dynamic ESM workflow (without pruning mechanism enabled) CPU usage on MN4.

- **AR:** Accuracy of the results represents the accuracy of the results obtained through the workflow implemented during the project compared to the standard procedures currently used. None of the approaches used in the dynamic ESM model are intended to reduce accuracy of the simulations unlike some other workflows in eflows4HPC where reduced precision is used. Hence, we consider this 100 percent.

4.4 External Evaluation

The external evaluation procedure involved representatives from Max-Planck-Institute for Meteorology (MPI-M) and German Climate Computing Centre (DKRZ). From the MPI-M we were evaluated by Dr. Lukas Kluft, who has more than 10 years of experience in running very high resolution climate models, processing of petabyte-scale climate data, development of climate model I/O, and data pre and post-processing software. From the DKRZ we were evaluated by Dr. Florian Ziemen, who has more than 10 years of experience in running several climate models of different complexity, managing petabyte-scale data and developing open source software for efficient data processing. Both of them are currently involved in projects that develop and run kilometer scale climate models funded by EU (EERIE, DestinE, nextGEMS) and BMBF (WarmWorld).

Both Dr Ziemen and Dr. Kluft positively evaluated both the Dynamic ESM workflow and additional functionality developed for ESM simulations in the framework of the eFlows4HPC project. They praise integration of in-memory data analysis climate modeling, which significantly enhances the simulation capabilities. Moreover, they note that the project shows successful collaboration across AWI and BSC, highlighting how this synergy fostered the development of cutting-edge tools and methodologies. Both experts agreed that the project outcomes meet the initial objectives set in the beginning of the project.

Dr. Kluft and Dr. Ziemen provided valuable insights and recommendations for enhancing the project's capabilities. Specifically, they discussed the potential for refining the model's response to critical events such as cyclones. Dr. Ziemen suggested further exploring use of the Zarr format to optimize data handling. Additionally, they both highlighted the importance of incorporating fault tolerance and resilience in the system. These suggestions are aimed at advancing the project's efficiency in data processing and enhancing its capabilities.

5 Analysis and Feature Extraction Validation

5.1 Workflow Use Case Applications

The statistical analysis and feature extraction workflow of Pillar II exploits the eFlows4HPC software stack for the integration of the ESM model execution and HPDA and ML approaches in order to analyze extreme events starting on the output of the ESM simulations. Specifically, PyCOMPSs allows to orchestrate different tasks of the workflow synchronizing the different tools and algorithms runs (e.g, Ophidia, Tstorms, ML for Tropical Cyclone, etc.). The following figure shows a simplified schema of the main blocks of the workflow.

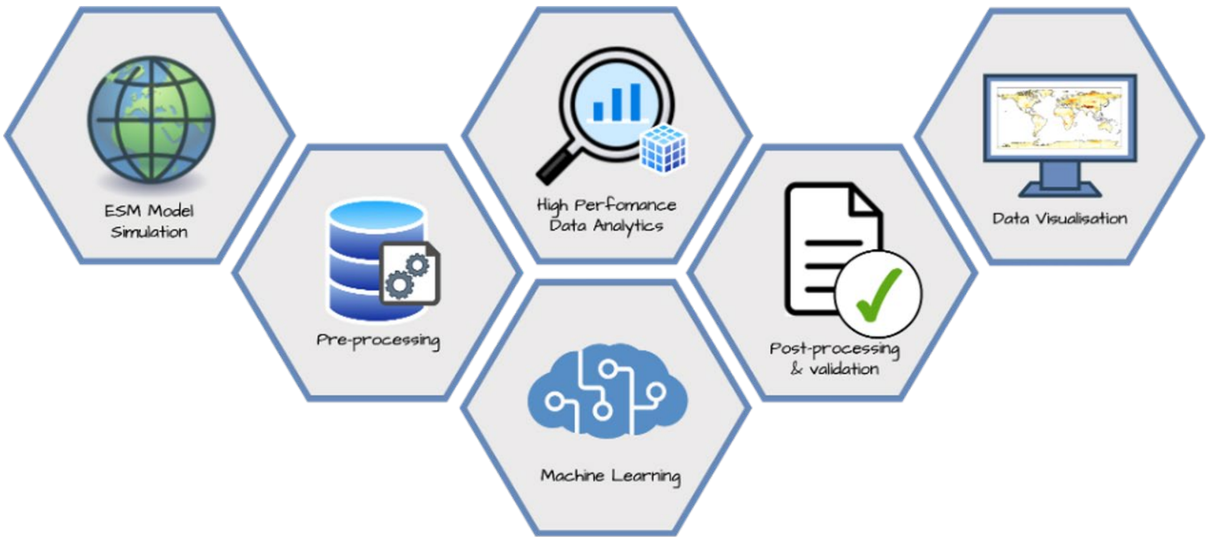


Figure 16. Analysis and feature extraction workflow schema.

The CMCC-CM3 simulation component marks the initial stage within the overarching framework. Usual configuration runs on 936 cores on Zeus Supercomputer at CMCC (52 full compute nodes) generating NetCDF files on a daily basis (each approximately 271 MB in size). The subsequent steps of the workflow occur once the working directory contains files spanning an entire year (amounting to nearly 100 GB). Facilitating concurrent execution of both ESM simulation and post-processing is essential, given that the climate simulation may extend over multiple decades (e.g., 30-35 years) and necessitate several days (or even months) to complete, depending on the HPC infrastructure employed. As mentioned above, to enable concurrent task execution, PyCOMPSs is used; more in detail, it employs a streaming interface, which monitors file production progress and identifies when a complete new year's of data becomes available.

Once a full year of CMCC-CM3 output is available, PyCOMPSs activate the data analytics operators to compute the extreme climate event indices for heat waves and cold spells. Given the large volume of data involved, High-Performance Data Analytics (HPDA) operators from the Ophidia framework are employed to process and aggregate the datasets in parallel. This involves handling multiple datacubes comprising nearly 1.3 billion data points for extreme event computations. Additionally, leveraging Ophidia's capability to store datasets in memory between different operator executions, baseline value files containing long-term historical averages (totaling around 2.6GB) are loaded only once and utilized throughout the workflows for the indexes computation, thereby reducing the need for recurring read operations from the storage.

Tropical Cyclones (TCs) are intricate phenomena driven by a blend of atmospheric and oceanic processes. Earth System Models facilitate the simulation of these intricate interactions, offering valuable insights into their formation, intensification, and trajectories. Nonetheless, identifying such extremes within extensive climate datasets remains arduous, primarily due to the large volume of data and the constraints of conventional detection methods. Machine learning (ML) techniques can aid in extracting significant spatial features associated with TC presence in gridded climate data.

The ML-enabled TC localization method deployed in eFlows4HPC enables the detection of TC presence based on a set of input climate variables simulated by ESMs (e.g., temperature, sea level pressure, wind speed, vorticity), as well as the localization of its center in terms of geographical coordinates. A Convolutional Neural Network (CNN) built on the Visual Geometry Group (VGG) architecture, previously trained on historical data, is employed to pinpoint TC centers. The sub-workflow encompasses five distinct tasks: the first four are carried out for each day of data, while the last one consolidates the results for an entire year of data. Specifically, the tasks involve: (i) post-processing of model simulations (e.g., regridding the CMCC-CM3 file), (ii) partitioning of data into non-overlapping patches and feature scaling, (iii) inference using pre-trained CNNs, (iv) geo-referencing the predicted TC center coordinates onto a global map, and (v) aggregation of daily outcomes into a unified NetCDF file for the entire year. Figure 17 shows that process subdivided in data collection, classification and localization steps.

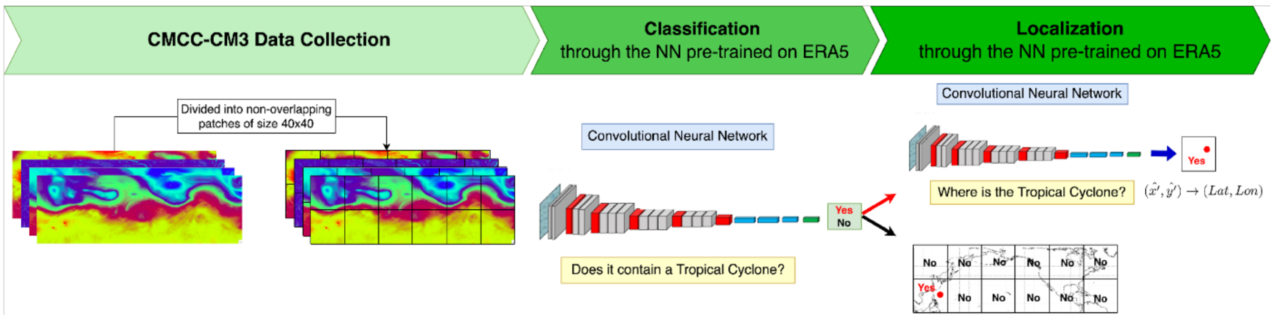

CMCC-CM3 Data Collection
Classification
through the NN pre-trained on ERA5
Localization
through the NN pre-trained on ERA5
Divided into non-overlapping patches of size 40x40
Convolutional Neural Network
Yes
No
Does it contain a Tropical Cyclone?
Convolutional Neural Network
Yes
Where is the Tropical Cyclone?
$(\hat{x}', \hat{y}') \rightarrow (Lat, Lon)$

Figure 17. Inference steps of the ML based approach for the TC detection.

Furthermore, the workflow executes deterministic TC tracking schemes, leveraging the TSTORMS software, to further validate the findings. This stage involves three tasks: (i) parallel aggregation of daily files into monthly files, (ii) parallel processing of the twelve monthly files containing 6-hourly atmospheric variables with TSTORMS to identify points exhibiting TC characteristics, and (iii) analysis of the files generated in the previous step in conjunction with TSTORMS to trace potential TC points and extract TC trajectories. The PyCOMPs streaming interface is also utilized in this scenario to monitor file production progress and detect the availability of the twelve monthly files.

The case study was conducted on a geographically distributed infrastructure, utilizing the HPCWaaS interface hosted on a service node located at the Barcelona Supercomputing Center (BSC) in Barcelona, Spain. Meanwhile, the entire workflow was carried out on the Zeus supercomputer situated at CMCC in Lecce, Italy. The Alien4Cloud service, operating at BSC, facilitated the deployment of the environment onto the Zeus CMCC cluster and the submission of the workflow. This was achieved by remotely interacting with the cluster scheduling system, exploiting the configuration outlined in the TOSCA topology. PyCOMPSs coordinated the execution of individual workflow tasks on Zeus.

A further test was deployed using two different HPC architectures: Zeus at CMCC and Nord3 at BSC. In particular, the CMCC-CM3 model is run on Zeus at CMCC. At the end of the production of each year of simulation, the output is transferred (the python script ssh2ssh is used) on the Nord3 machine to the BSC where the workflow continues with the pipeline relating to the analytics and TC detection phase. A4C drives the stage-in of the necessary inputs at the two target infrastructures while PyCOMPSs the distributed execution of the two different workflow phases. Figure 18 shows the different modules and where they are executed.

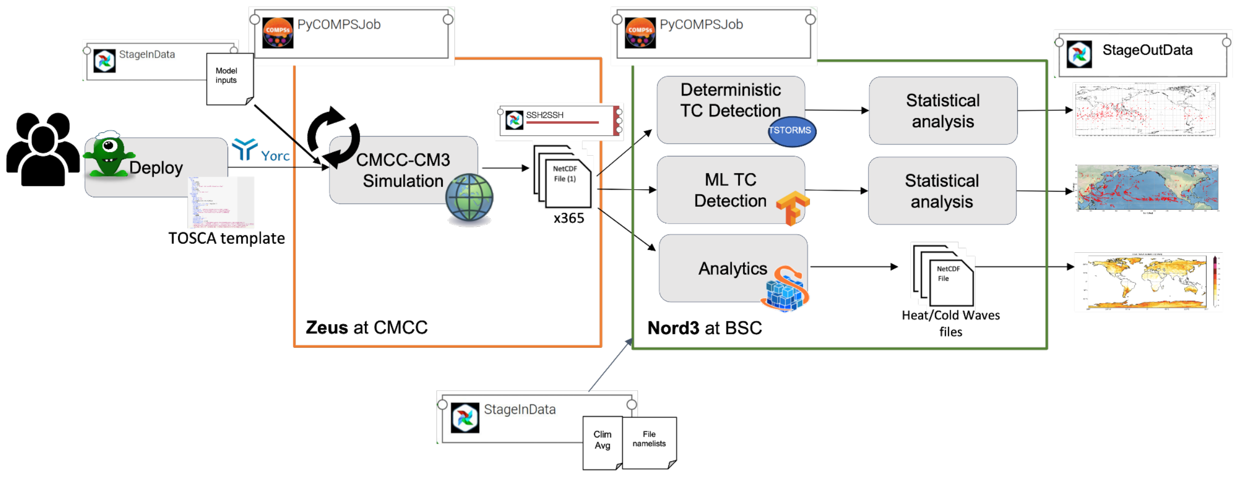


Figure 18. Analysis and feature extraction workflow distributed at CMCC and BSC.

5.2 Requirements validation

Table 5. Analysis and feature extraction workflow requirements validation

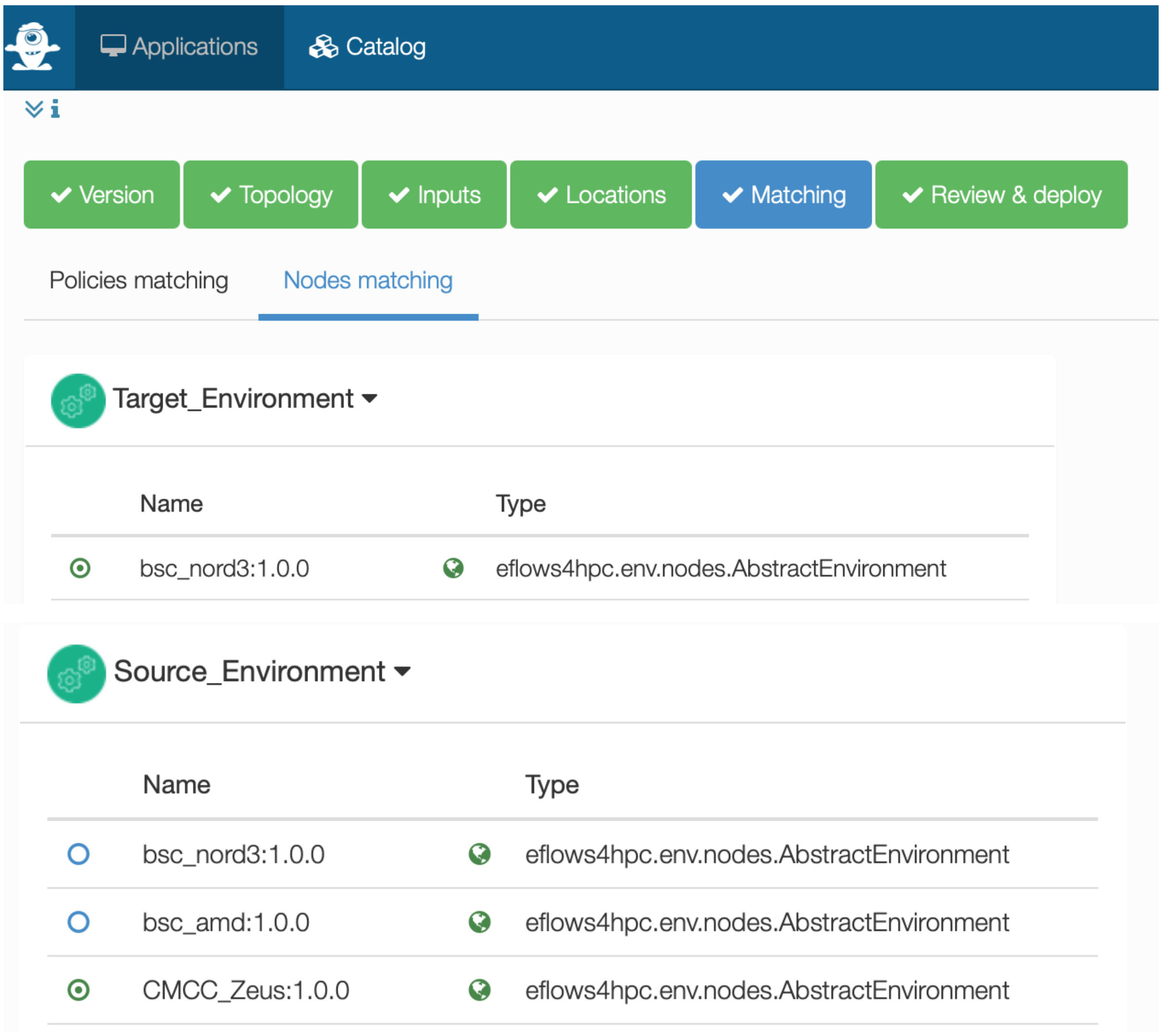
ID	Name	Description	Priority	Status
1	Execution Robustness	Management of fault tolerance during the workflow execution including checkpoints or retries. For example, during a large execution if a node fails, the workflow must be able to recover and continue to the end.	Should	Using PyCOMPSs and CMCC-CM3 resubmission capability
2	Portability	Workflow components should be portable to various types of HPC infrastructures.	Should	Docker/Singularity containers
3	Integrated workflow manageme nt	Requires the Management of task dependencies, execution of parallel simulations on different HPC infrastructures, management of batch jobs (submission, monitoring, cancellation), management of conditional paths in a transparent way.	Must	By PyCOMPSs capabilities
4	Integration with long-term archive/rep ository storage	Results may be stored in long-term storage for archiving purposes, second use (e.g. downstream services) and/or to satisfy FAIRness policies.	May	Using DLS capabilities on B2SHARE

5	Workflow adaptability	Capability to easily manage, cancel, replace and add components invocations in the workflow, for instance allowing the execution starting from the n-th step.	Should	Simplified version avoid the CMCC-CM3 model run + internal computation of climatological mean
6	Access to intermediate in-memory results	The workflow should be able to retrieve data/intermediate outputs of the running processes directly from memory.	Must	Exploiting Ophidia/xarray in memory storage
7	ML/DL capabilities	Requires the support for training and inference of Neural Network models for example for Tropical Cyclone detection.	Must	ML approach for TC detection
8	DA capabilities	Support for descriptive analytics (e.g., statistical analysis) exploiting fast in-memory analysis.	Must	Exploiting Ophidia/xarray in memory analysis capabilities
9	High Performance Computing support	Climate models have to be executed on computing infrastructures capable of providing a large amount of processing and memory resources.	Must	Run allowed on Zeus/Nord3
10	Multi-member analysis	Support for concurrent execution of sub-workflows starting from different inputs (configurable) and comparison of the sub-workflows results.	Must	CMCC-CM3 run configurable and executions outputs comparable (e.g., different reference years)

In the following the different requirements are evaluated.

- **Execution robustness:** This requirement is met through the use of PyCOMPSs and the checkpointing embedded into the CMCC-CM3 model. In the first case, PyCOMPSs allows to re-execute the failed task by detecting the process exit code. This allows re-execution of the task and continuation of the workflow without re-executing the entire workflow. In the second case, the CMCC-CM3 model exploits an internal checkpointing mechanism; through this setting (which can be enabled or disabled when submitting the run) the model is able to save its state at regular intervals using a set of files called restart files. In the event that the execution were to fail, the implemented workflow uses PyCOMPSs for the retry and restart files to continue the simulation starting from the time step that caused the fault.
- **Portability:** This requirement can be satisfied through the containerization of the workflow, the tools and applications necessary for its execution and the HPCWaaS service

offered by eFlows4HPC or by providing standard installation procedures exploiting well-known and supported packages. More specifically, some components of the workflow have been containerized using docker and singularity which allows portability across different HPC infrastructures. The use of Spack also facilitates the configuration procedure of the various modules while the A4C framework associated with Yorc allows deployment and subsequent execution on the identified target architecture. Concerning the remaining components, they rely on common packages and libraries, available for the main infrastructures and O.S. Figure 19 shows the A4C interface where it is possible to select the execution target environments.



Target_Environment

	Name		Type
●	bsc_nord3:1.0.0		eflows4hpc.env.nodes.AbstractEnvironment

Source_Environment

	Name		Type
○	bsc_nord3:1.0.0		eflows4hpc.env.nodes.AbstractEnvironment
○	bsc_amd:1.0.0		eflows4hpc.env.nodes.AbstractEnvironment
●	CMCC_Zeus:1.0.0		eflows4hpc.env.nodes.AbstractEnvironment

Figure 19. A4C interface for the selection of the target environments.

- **Integrated workflow management:** This requirement is satisfied through the use of the software stack provided by eFlows4HPC and in particular by exploiting the capabilities offered by PYCOMPSs. In fact, PYCOMPSs is able to manage the entire workflow in terms of dependencies between tasks both about the sequence of operations to be performed (task dependency) and with regards to the dependency of the inputs and outputs (data dependency). For example, in the first case, two tasks that must be executed sequentially belong to two different and sequential statements in their PYCOMPSs definition

```
@task(returns=object, movingavgcubes=COLLECTION_IN)
def OphClimAvg(client, input_path, movingavgcubes, filename):
    cube.Cube.client = client
    baseline=cube.Cube.intercube2(cubes=movingavgcubes, ncores=4*NHOST)
    baseline.exportnc2(output_path=input_path,output_name=filename)
    return baseline

@task(datacubes=COLLECTION_IN, dependences=COLLECTION_IN)
def OphDelete(client, datacubes, dependences):
    cube.Cube.client = client
    for x in range(len(datacubes)):
        datacubes[x].delete()

@task(returns=object, files=COLLECTION_IN)
def OphImport(client, input_path, files, measure, op):
    cube.Cube.client = client
    #REAL CASE: src_path = files
    src_path = join(input_path, 'CMCC-CM3/2000_cam6-nemo4_025deg_tc.cam.h1.0030-*-*-00000.nc')
    Year = cube.Cube.importncs(src_path= src_path,
        container='Test',
        measure=measure,
        import_metadata='yes',
        nfrag=4,
        nthreads=4*NHOST,
        ncores=1,
        nhost=NHOST,
        imp_dim='time',
        imp_concept_level='6', vocabulary='CF',hierarchy='oph_base|oph_base|oph_time',
        description='6-Hours Temps'
        )
    # Computation of the maximum or minimum on the 6-hours values to obtain a value per day
    dailySeries = Year.apply(
    query="oph_reduce2('OPH_FLOAT', 'OPH_FLOAT', measure," + op + ",4)", nthreads=4*NHOST)
    Year.delete()
    return dailySeries
```

An example of the second case, however, can be identified in the functionalities provided by the Streamin function of PYCOMPSs, used within the workflow: in this case an operation defined through a task is delayed until the proper inputs are available, i.e. until the previous task produced the necessary outputs. Within the workflow this function is used to properly synchronize the production of the necessary NetCDF files by the CMCC-CM3 model and the subsequent heat/cold waves indices computation and tropical cyclone detection operations.

```
@task(fds=STREAM_IN, returns=list)
def read_files(fds):
    num_total = 0
    list_files = []
    #REAL CASE: while num_total < 365:
    #TEST with a model that produces five days
    while num_total < 5:
        # Poll new files
        print("Polling files")
        new_files = fds.poll()
        # Process files
        for nf in new_files:
            print("RECEIVED FILE: " + str(nf))
            if str(nf).endswith('.nc'):
                if str(nf).split('/')[-1].split('.')[2] == 'h1':
                    list_files.append(str(nf))
        # Sort read files
        list_files = sorted(list_files)[ : -1]
        # Accumulate read files
        num_total = len(list_files)
        # Sleep between requests
        time.sleep(SLEEP)
    # Return the number of processed files
    return list_files
```

Furthermore, the management of batch job submission, as well as monitoring and their cancellation, is managed at a high level via A4C which allows to start the run of the workflow, monitor its status and possibly stop its execution.

- **Workflow adaptability:** This requirement concerns the possibility of managing, eliminating, replacing or adding components invocations in the workflow. This possibility is satisfied by the modularity with which the workflow was designed and by the conditional paths inserted into it. An example is given by the calculation of the climatological mean, necessary for the calculation of heat waves and cold spells; in fact, in this case, the workflow is set up to act in two distinct modes. The first method concerns the loading of these values from a pre-produced file: in this case, the corresponding climatological mean values have previously been saved to a file and the ingestion phase in Ophidia occurs only once. In the second case, however, Ophidia is able to calculate the climatological mean from the complete set of annual files (a total of 20 years) produced by the model; in this case, the process is obviously slower due to the large amount of data to be processed (about 2 TB).

```
# CLIMATOLOGICAL MEAN COMPUTATION FOR TSMX
print("[LOG] CLIMATOLOGICAL MEAN IMPORT")
if(compss_file_exists(maxBaseline)):
    print("Exists")
    clim_avg_tsmx = OphImportClimAvg(cube.Cube.client, inputPath, 'max_clim_avg.nc', 'TSMX')
else:
    maxdatacubes = [0 for i in range(NYEARS)]
    for i in range(NYEARS):
        maxdatacubes[i] = OphImportForBaseline(cube.Cube.client, inputPath, str(i+12).zfill(4), 'TSMX', "'OPH_MAX'")
    clim_avg_tsmx = OphClimAvg(cube.Cube.client, inputPath, maxdatacubes, 'max_clim_avg')
```

To speed up this process, however, a new functionality has been implemented in eFlows4HPC for the direct calculation of the climatological mean vs the previous mechanism which used multiple operations in sequence.

```
@task(returns=object, movingavgcubes=COLLECTION_IN)
def OphClimAvg(client, input_path, movingavgcubes, filename):
    cube.Cube.client = client
    baseline=cube.Cube.intercube2(cubes=movingavgcubes, ncores=4*NHOST)
    baseline.exportnc2(output_path=input_path,output_name=filename)
    return baseline
```

- **Access to intermediate in-memory results:** This requirement is met by using the Python Xarray library and Ophidia. Both, in fact, allow keeping the data in memory for subsequent processing without necessarily having to download it to storage. In particular, xarray uses the Dataframe in-memory data format that can be shared between subsequent instructions in order to avoid saving to disk.

```
@task(returns=list, vars=COLLECTION_IN)
def get_dataset_patches(ds, vars, patch_size):
    # patch the dataset
    patch_ds = ds.coarsen({'lat':patch_size, 'lon':patch_size}, boundary="trim").construct({
lon':("cols", "lon_range"), 'lat':("rows", "lat_range")})
```

Ophidia, on the other hand, natively enables in-memory analysis, keeping in memory the datacubes needed for subsequent analyzes. Within the implemented workflow, this is for example the case of the climatological mean which is loaded into Ophidia's in-memory storage only once and then used for each year of simulation produced by the CMCC-CM3 model for the calculation of the heat wavers and cold spells. Furthermore, Ophidia allows distributing the information related to a datacube across the memory of multiple computing nodes, allowing on the one hand the possibility of using a larger memory space and, on the other, processing the information in memory in parallel using multiple involved tasks.

- **ML/DL capabilities:** In the implemented workflow, the tropical cyclone detection phase follows an approach designed in eFlows4HPC and based on Machine Learning techniques. Through the use of PyCOMPSs, the execution of the algorithm is inserted as a task within the overall workflow, before the production of the final maps and following the run of the CMCC-CM3 model, in parallel with the run of the operations performed by Ophidia for the calculation of heat/cold waves.
- **DA capabilities:** This requirement concerns the support for descriptive analytics exploiting fast in-memory analysis. This requirement is satisfied by using the xarray library and Ophidia. Both, in fact, allow you to carry out statistical analysis operations in memory, applied, for example, via Ophidia for the calculation of heat/cold waves indices. Furthermore, as previously mentioned, Ophidia is able to use its data analytics functions by also taking advantage of the parallelism offered by an HPC infrastructure.
- **High Performance Computing support:** This requirement concerns climate models that need computing infrastructures capable of providing a large amount of processing and memory resources to be executed. It is satisfied because the model used (CMCC-CM3 coupled model) is executed on HPC infrastructure. In more detail, it requires 936 cores for its execution, 540 for its atmospheric component, 396 for its oceanic component. Figure 20, shows a schema of the CMCC-CM3 coupled model along with the different components.

  Furthermore, the entire workflow is performed on HPC infrastructure, not only the climate model but also the analysis of extreme events (tropical cyclones detection and extreme events indices computation).
- Multi-member analysis: This requirement concerns the possibility of supporting concurrent executions of sub-workflows starting from different inputs in a configurable manner and the possible comparison of the results related to the sub-workflows. This requirement is satisfied through the use of the software stack provided by eFlows4HPC, the architectural design and the internal properties of the software/tools used. More in detail, the CMCC-CM3 model is configurable by modifying the input parameters; in particular it is possible to change the input files (forcings) and the initial configuration. The workflow design combined with the HPCWaaS architecture allows performing concurrent runs of the model and comparing the outputs that are stored in the user's personal repository.

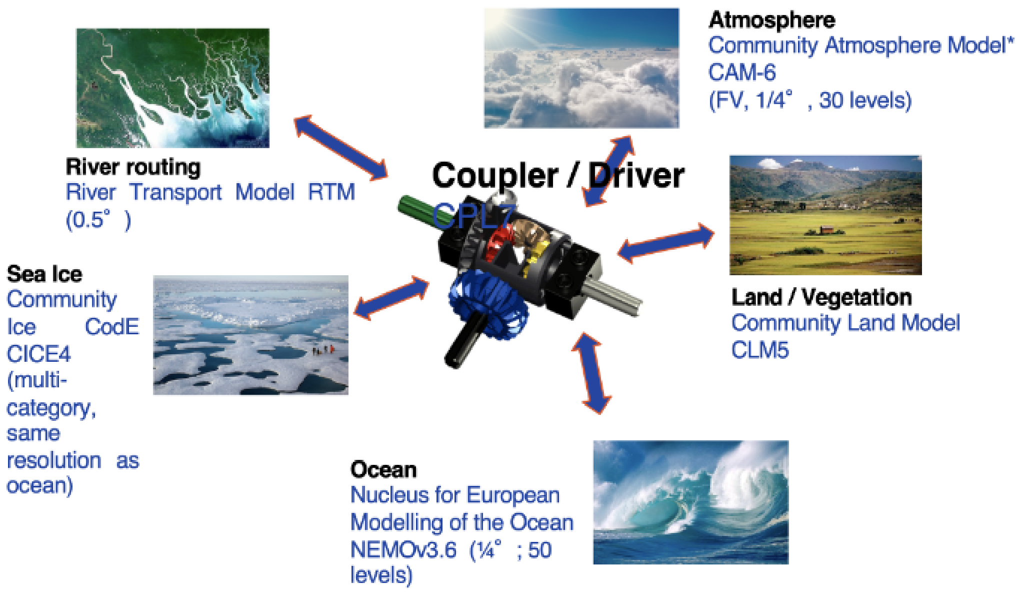

Atmosphere
Community Atmosphere Model*
CAM-6
(FV, 1/4° , 30 levels)
River routing
River Transport Model RTM
(0.5°)
Coupler / Driver
CPL7
Land / Vegetation
Community Land Model
CLM5
Sea Ice
Community Ice CodE
CICE4
(multi-category, same resolution as ocean)
Ocean
Nucleus for European Modelling of the Ocean
NEMOv3.6 (¼° ; 50 levels)

Figure 20. Components of the CMCC-CM3 coupled model.

5.3 Metrics evaluation

Table 6. Analysis and feature extraction workflow metrics evaluation

Acronym	Name	Description	Value
LoC	Lines of Code	Number of Lines of code in the workflow implementation.	450 (Python code)
DoP	Degree of Portability	Percentage of workflow components that can be reused in other infrastructures and workflows.	80%
DT	Deployment time	Time elapsed to deploy the workflow	38 seconds
ET	Execution Time	Time elapsed to execute a workflow.	341.2 seconds
SU	Speed-up	Execution time improvement when running with larger resources.	Scaling up the number of computing nodes from 2 to 32: 68.4% in the worst case
Eff	Efficiency	Execution time degradation when running larger problems.	Scaling up the number of computing nodes and input data: 69% in the worst case

IOT	I/O Time	Percentage of Execution time performing I/O operations.	84.72%
FTC	Fault-tolerant components	Percentage of workflow components that are fault-tolerant.	100% (PyCOMPSs allows the rerun of the wf tasks)
CH	Core/Hour	Number hours of a CPU Core consumed by the workflow execution.	~ 6400 CPU/hours + 12 GPU/hours
EC	Energy Consumption	Energy consumed (Wh or Joules) associated with a workflow execution.	140.4 W
AR	Accuracy of the results	Accuracy of scientific results should not degrade.	95.36%

- The **LoC** are calculated considering the PyCOMPSs workflow from the submission to the final extraction of the maps and results. Compared to the first project phase, the number of lines of code was reduced from 949 to 450 thanks to a series of code optimizations and a more extensive use of PyCOMPSs.
- The **DoP** represents the percentage of the number of workflow components that can be reused in other infrastructures. The CMCC-CM3 model represents a component that is not portable to other infrastructures. As regards the remaining part of the workflow, the individual components are portable on different infrastructures: Ophidia has been containerized and is available as a docker and singularity image, the ML algorithm requires standard Python libraries (such as tensorFlow and xarray), while the tool TSTORMs is third-party software and has a standard installation in a Linux environment. Overall, therefore, we can estimate that the portable percentage of the workflow is equal to 80%.
- The **DT** represents the time needed to deploy the workflow. For the analysis and feature extraction workflow it represents the time necessary to activate the run of the CMCC-CM3 model, corresponding to the first task of the workflow, on the target infrastructure (Zeus at CMCC) via the interface provided by A4C, approximately 38 seconds.
- The **ET** represents the execution time for executing the entire workflow. The total time for the execution of the workflow is equal to 341.2 seconds. It is important to underline that, if we compare the execution time at the beginning of the project, it was reduced from over 18 minutes to just under 6 minutes (with the same number of cores and temporal timesteps analyzed). This optimization was obtained through the new implementations carried out in eFlows4HPC (for example the new procedure for calculating the climatological average). As regards the ML algorithm for the detection of tropical cyclones, the inference phase is part of the overall workflow and is included in the total workflow time; however, as regards the training phase, the training time has been reduced from over five hours to just under three using a better memory and I/O management.

- **SU** represents the improvement in execution time when the workflow is executed using more computing resources. The speed up was calculated considering the analysis pipeline, including the calculation of heat waves and cold spells indices and the detection of TCs (in fact, the CMCC-CM3 model runs with a fixed number of cores per configuration used). Figure 21 shows the speed-up calculated as the computational resources used increase. The test involved up to 32 computing nodes of the Zeus supercomputer at CMCC, corresponding to 1152 cores used and was performed considering the run of the analysis pipeline on the output of one year of the CMCC-CM3 model, corresponding to 98.92 GB.

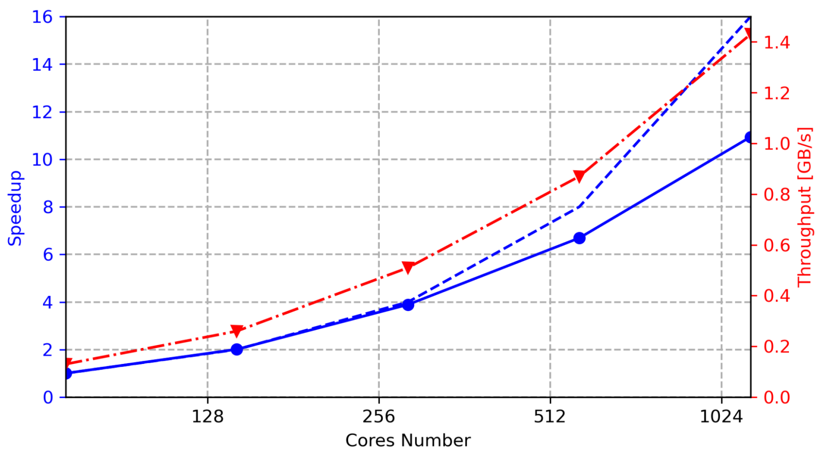


Figure 21. Speed-up of the analysis and feature extraction workflow.

- **Eff** is an index that represents the degradation of computation time when executing larger problems. As in the previous case and for the same reasons the calculation was performed considering the analysis pipeline. In this case (Figure 22), the test was performed using up to 32 nodes of the Zeus supercomputer and varying the input data from 6.18 GB up to 98.92 GB.

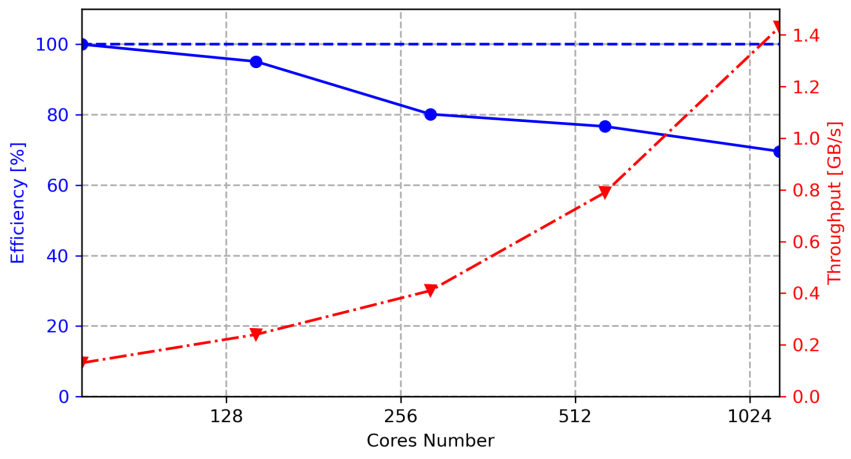


Figure 22. Efficiency of the analysis and feature extraction workflow.

- **IoT** represents the percentage of execution time dedicated to IO operations. In climate analyses, the IO phase usually covers a large part of the execution time as the amount of data being read and written is very high. The CMCC-CM3 model, for example, has a production rate of 0.36 SYPD (0.36 years of simulated products per day of execution) and every year produces datasets of 100GB in size (without considering the accessory and restart files). The post-processing and data analytics phase involved in eFlows4HPC starts from these datasets to start its execution. In eFlows4HPC optimizations have been used and implemented for the reduction of I/O time which still covers 84.72% of the total execution time.
- **FTC** represents the percentage of workflow components that are fault-tolerant. As already mentioned in the requirements description, the fault tolerance is fulfilled by utilizing PyCOMPSs alongside the embedded checkpointing feature within the CMCC-CM3 model. PyCOMPSs detects the exit code of the failed task, enabling its re-execution without restarting the entire process; the CMCC-CM3 model includes an internal checkpointing mechanism by saving its state using a set of restart files. So we can consider the percentage of FTC equal to 100%.
- **CH** represents the number of hours of a CPU Core consumed by the workflow execution. For this metric we considered the entire workflow, including the run of the CMCC-CM3 model and subsequent pipeline related to the calculation of heat/cold waves and TC detection. In total the workflow uses approximately 6400 cores/hours. Furthermore, the training phase of the ML model takes 12 GPU/Hours
- **EC** represents the energy consumed associated with a workflow execution. This test was carried out considering the energy used on 3 computing nodes of the Zeus supercomputer during the analytics pipeline on the outputs of the CMCC-CM3 model related to a year of simulation. The energy consumed by the process corresponds to about 140.4W
- **AR** represents the accuracy of the results obtained through the workflow implemented during the project compared to the standard procedures currently used. In particular, the machine learning process for the detection of tropical cyclones was taken into consideration for the analysis, comparing the results with the TSTORMs tool, currently considered the de facto standard for this type of analysis. The accuracy of the ML process itself is at 95.36% regarding the ML test phase, Figure 23 shows a comparison map between the output of the algorithm implemented in eFlows4HPC and the TSTORMs tool.

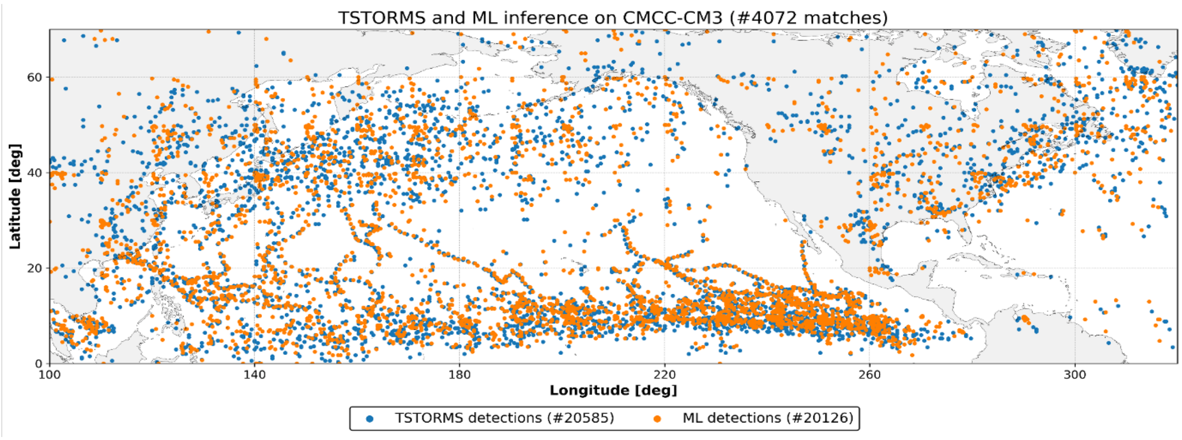

TSTORMS and ML inference on CMCC-CM3 (#4072 matches)
Latitude [deg]
0
20
40
60
100
140
180
220
260
300
Longitude [deg]
● TSTORMS detections (#20585)
● ML detections (#20126)

Figure 23. Comparison between the TSTORMS and the ML approach results.

5.4 External evaluation

The external evaluation procedure involved the CMCC staff belonging to the project and Dr. Stefano Natali, co-founder, managing director and space business manager of MEEO Srl (Italy) and SISTEMA GmbH (Austria). He has more than 20 years' experience in the analysis of satellite data, for atmospheric and biophysical parameters retrieval. He has more than 15 years' experience in project management in European Commission, European Space Agency and national projects and he is fully involved in the development and adoption of the Advanced geospatial DAta Management platform (ADAM) to facilitate the access and exploitation of a large variety and volume of geospatial data.

Dr. Natali positively evaluated the work carried out in the eFlows4HPC project, judging the project and case study to be very interesting and ambitious and the architecture well structured. He also positively assessed that all the requirements presented at the beginning of the project had been satisfied.

Finally, he gave a series of useful feedback regarding the possibilities of extending and optimizing the case study and the applied procedures. In particular, regarding the ML procedure applied, he suggested verifying the adoption of other climate models than those currently used (ERA5 for training and CMCC-CM3 for inference). This could reduce the number of false positives extracted by both the ML procedure and the TSTORMS tool. He also suggested applying the heat waves and cold spells indices computation procedure on a larger dataset in order to reduce the impact of the overhead.

6 Conclusions and recommendations

The two workflows implemented in the context of Pillar II - Dynamic and adaptive workflows for climate modeling, highlight the effectiveness of the application of the HPCWaaS paradigm and the software stack provided by the eFlows4HPC project in the context of climate research applications. The requirements defined at the beginning of the project were satisfied and the metrics demonstrated the effectiveness in the implementation of the two workflows. Furthermore, external evaluations demonstrated the correct implementation and applicability in the scientific context, reporting useful advice and considerations for future developments.

As general recommendations and future developments we can take the following into consideration:

- Alien4Cloud will not be further maintained, so the entry point of the end-to-end workflows would need to be ported to some other solution. In this sense, the experience acquired, the decoupling of the pipelines executed on the target machines (driven by PyCOMPSs) from the submission mode (driven by A4C and Yorc), along with the TOSCA configuration that could be re-used in other platforms (such as LEXIS[2], a EU platform maintained by IT4Innovations (CZ)) will make straightforward the migration to other solutions;
- It would be possible to evaluate other workflow management solutions, such as Cylc[3], ecFlow[4] or Autosubmit[5], more commonly used in the climate context, with some adaptations to the control mode of the ensemble members which would make the Dynamic ESM workflow more portable;

- Despite demonstrating potential of In-memory analysis with Cassandra as backend, at least in terms of saving storage space and dynamic workflows, it may still need fine tuning for effectiveness and evaluate it against higher resolution simulations. It is also interesting to compare using other databases such as Redis as IO backends.
- The statistical analysis and feature extraction workflow could be applied starting from the products of other climate models or in other contexts. In the first case, higher horizontal resolution datasets could be used. In the second case, ocean models could be used in order to calculate indices relating to marine heat waves or for the detection of marine eddies.
- The statistical analysis and feature extraction workflow could be embedded within the CMCC-CM3 model in order to directly access the data produced even before they are saved to disk. There are specific solutions (e.g. XIOS[6]) coupled with the climate model useful for extracting various types of statistical information but they are linked to the model they belong to and there is a lack of a general purpose solution that can be applied to multiple models or to models that belong to a certain context (e.g., ocean models).

7 Acronyms and Abbreviations

- ESM: Earth System Model
- HPC: High-Performance Computing
- MN4: MareNostrum 4
- TOSCA: Topology and Orchestration Specification for Cloud Application

8 List of figures and tables

9 References

[1] Bradner, S. O. (1997). Key words for use in RFCs to Indicate Requirement Levels. RFC 2119. Request for Comments. https://rfc-editor.org/rfc/rfc2119.txt

[2] LEXIS Project: https://lexis-project.eu/web/

[3] Cylc: https://cylc.github.io/

[4] ecFlow: https://ecflow.readthedocs.io/en/latest/index.html

[5] Autosubmit: https://autosubmit.readthedocs.io/en/master/

[6] XIOS: http://forge.ipsl.jussieu.fr/ioserver

[7] Xarray: https://docs.xarray.dev/en/stable/

[8] Scalability and some optimization of the Finite-volumE Sea ice–Ocean Model, Version 2.0 (FESOM2), Koldunov et al. 2019, https://doi.org/10.5194/gmd-12-3991-2019

[9] Dask: https://docs.dask.org/en/stable/

9.1 Software Repositories

Name of Software	***Description, link***
COMPSs	*COMPSs encapsulates PyCOMPSs that is used to orchestrate the workflow tasks.* *https://github.com/bsc-wdc/compss*
Eflows4HPC software Catalog	*This repository contains specifications to build softwares used in workflow either in containers or directly on a HPC.* *https://github.com/eflows4hpc/software-catalog*
WORKFLOW REGISTRY	*This repository contains all the workflows across Eflows4HPC WPs. Pillar_II directory contains workflows for this Pillar.* *https://github.com/eflows4hpc/workflow-registry*
HECUBA	*HECUBA provides an efficient API to store and retrieve data for in-memory analysis for Dynamic ESM workflows.* *https://github.com/bsc-dd/hecuba*
FESOM2	*FESOM2 is the ocean-climate model used in Dynamic ESM workflows. Following branch contains modifications to its source with HECUBA/CASSANDRA as IO-backend and for workflows.* *https://github.com/FESOM/fesom2/tree/eflows_hecuba_templates_update*

OPHIDIA	*OPHIDIA provides an efficient API to perform in-memory analysis for Analysis and Feature Extraction workflows.* *https://github.com/OphidiaBigData*