

HORIZON-EUROHPC-JU-2021-COE-01

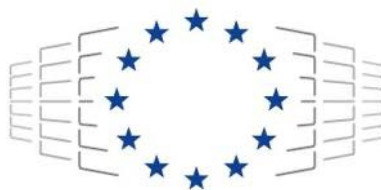


**The European Centre of Excellence for Engineering
Applications**

Project Number: 101092621

D4.2

**First Updated Workflows for engineering simulations
progress report**



The EXCELLERAT P2 project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101092621. The JU receives support from the European Union’s Horizon Europe research and innovation programme and Germany, Italy, Slovenia, Spain, Sweden and France.

Work Package:	4	Workflow Development
Author(s):	Tuan Anh Dao	KTH
	Antoine Dauplain	CERFACS
	Dennis Grieger	USTUTT
	Eman Bagheri	FAU
	Adalberto Perez	KTH
	Vaishali Ravishankar	FGH
	Matic Brank	UL
	Francesco Salvatore	CINECA
Approved by	Executive Centre Management	24.06.2025
Reviewer	Jernej Kovačič	UL
Reviewer	Marvin Hubl	SSC
Dissemination Level	Public	

Date	Author	Comments	Version	Status
2025-05-20	Antoine Dauplain, Dennis Grieger, Eman Bagheri, Adalberto Perez, Vaishali Ravishankar, Matic Brank, Francesco Salvatore	Full document drafting	V0.1	Draft
2025-05-27	Tuan Anh Dao	Document revision and finalisation	V1.0	Final
2025-06-12	Tuan Anh Dao	Implementation of first internal review	V1.1	Final

2025-06-25	Tuan Anh Dao	Implementation of the suggestions from the project manager	V1.2	Final
------------	--------------	--	------	-------

List of abbreviations

ABI	Application Binary Interface
ACF	Autocorrelation Function
API	Application Programming Interface
ASMR	Automatic Static Mesh Refinement
BO	Bayesian Optimizer
CFD	Computational Fluid Dynamics
CI	Continuous Integration
HPC	High-Performance Computing
IO	Input-Output
MPI	Message Passing Interface
PCA	Principal Component Analysis
SME	Small and Medium-sized Enterprises
SQL	Structured Query Language
SST	Scalable, Streaming Transport
UL	University of Ljubljana
VTK	Visualisation Toolkit
WP	Work Package

Executive Summary

This report presents the first updated workflows for engineering simulations developed in EXCELLERAT P2. The focus is on improving simulation workflows for High-Performance Computing (HPC), particularly in enabling in-situ techniques, supporting outer loop optimisation, and enhancing workflow usability through standardisation.

We describe progress in integrating in-situ visualisation tools with Computational Fluid Dynamics (CFD) codes like Alya and Neko; and demonstrate methods for acoustic analysis using non-blocking streaming. Developments in shape optimisation using Bayesian methods are also outlined, with early results from NACA airfoil studies. In parallel, new tools such as Lemmings and SCALES support automated, user-friendly workflows.

Further work includes initial results in Automatic Static Mesh Refinement (ASMR), showing generation of meshes up to 50 million cells, with the goal of creating billion-cell meshes for exascale applications. We also introduce a systematic way to collect job performance data from real production runs to better understand and tune workflow performance.

Table of Contents

1	Introduction	9
2	In-situ Techniques	9
2.1	Neko, UQit and AcoNeko	9
2.2	L2G, OpenFOAM, Raysect.....	11
2.3	SOD2D	12
2.4	Vistle	12
2.5	STREAmS.....	13
3	Outer loop optimisation.....	14
3.1	Optimisation Workflow.....	14
3.2	Results and Inference	15
3.3	Dimensionality Reduction.....	16
4	Homogenisation/Standardisation of HPC Workflows	17
4.1	Tools for Workflow Development	17
4.1.1	Lemmings.....	17
4.1.2	SCALES	18
4.2	Recommendations	19
4.2.1	Unified Chronological logging.....	19
4.2.2	A Unique Identifier for workflow instances	19
4.2.3	No File Movement	19
4.2.4	Version and Document the Workflow	19
4.2.5	A Sandbox Scheduler	20
4.2.6	A mock-up Solver	20
4.2.7	End User Commands.....	20
4.2.8	Container issues.....	21
4.2.9	Reduced version of workflow recommendations.....	21
4.3	Automatic static mesh refinement.....	22
4.4	Monitoring Applications with Job Data Points	23
5	Since D4.1: Progress Summary and Key Metrics	24
6	Conclusions	25
7	References	26

Table of Figures

Figure 1: The in-situ workflow for uncertainty quantification between Neko and UQit.....	10
Figure 2: Mean Streamwise component of velocity (top) and the variance of the sample mean estimator averaged over 60 convective time units	11
Figure 3: Velocity field of a Taylor Green Vortex obtained using Vistle and the SENSEI interface of Alya.....	13
Figure 4: Bezier curve representation of the airfoil with control points	15
Figure 5: Best airfoil profile generated by the Bayesian Optimizer (left), x-velocity (right, top) and y-velocity (right, bottom) results from CFD simulation of the airfoil	16
Figure 6: Worst airfoil profile generated by the Bayesian Optimizer (left), x-velocity (right, top) and y-velocity (right, bottom) results from CFD simulation of the airfoil	16
Figure 7: Plot of 2D PCA on input data	16
Figure 8: Results from DataViewer (left) and SimExplore (right)	17
Figure 9: The backbone loop of the Lemmings automation scheme. By injecting custom code into the orange blocks, the user can adapt the tool to most workflows.....	18
Figure 10: A screen capture of one of the high-level monitoring dashboards of SCALES. Workflows of tasks can be resubmitted on the cluster until satisfactory results are obtained. Unlike the Lemmings approach, all connections and expert commands are encapsulated behind this graphical dashboard	18
Figure 11: Figure showing how a mesh is automatically refined using mesh adaptation with the ASMR workflow. The configuration is an academic Hydrogen Burner (TUB burner). Our final goal is the generation of a 1 billion cell mesh in a single workflow	23

Table of Tables

Table 1: Summary of key changes and improvements since Deliverable D4.1 24

1 Introduction

High-Performance Computing (HPC) plays a critical role in enabling large-scale engineering simulations. As systems evolve toward exascale, the complexity and volume of simulation data increase significantly, introducing new challenges in how workflows are designed and executed. Traditional approaches, where simulations are performed in isolation from pre- and post-processing, no longer scale effectively. These approaches often depend on large-scale file Input-Output (IO) for storing and analysing results, which becomes a bottleneck at scale.

The goal of Work Package (WP) 4 is to address these challenges by developing workflows that support scalable, efficient, and reproducible simulations. This includes enabling in-situ analysis and visualisation, which avoids the need to store large intermediate data by embedding analysis directly in the simulation. Several use cases now include in-situ capabilities that allow data to be streamed and processed in memory, reducing both IO costs and time to insight.

Beyond in-situ methods, WP4 also looks at outer loop workflows, where multiple simulation runs are coordinated to achieve tasks like shape optimisation. These workflows are essential for design tasks in engineering, where results from simulations inform decisions on geometry or configuration. Automation and integration of tools such as solvers, mesh generators, and optimisers are central to making these workflows usable and efficient.

A key objective here is to make HPC workflows more accessible for industrial users, who often lack the deep HPC expertise found in research environments. To this end, the project continues to improve tools like Lemmings and SCALES, which simplify job management, workflow execution, and monitoring. These tools help abstract away the complexity of running jobs on different HPC systems, lowering the barrier to entry for Small and Medium-sized Enterprises (SMEs).

Another major challenge is mesh-generation, which remains a time-consuming and expertise-driven task in CFD. Manual mesh generation becomes impractical at exascale, so an ASMR pipeline has been developed. This workflow allows meshes to be iteratively refined based on simulation results, replacing manual tuning with a data-driven process. Early results have shown success in generating meshes with tens of millions of cells, with a long-term target of reaching a billion-cell mesh.

Finally, to ensure that workflows perform reliably across different environments and use cases, job monitoring strategies have also been introduced. By collecting consistent, structured metadata about each workflow execution, including input parameters, performance metrics, and hardware context, the project aims to build a foundation for robust performance analysis and optimisation.

Together, these developments mark an important step toward scalable, reusable, and user-friendly engineering simulation workflows designed for current and future HPC platforms.

2 In-situ Techniques

2.1 *Neko, UQit and AcoNeko*

To obtain uncertainty estimates without interrupting the core solver, we coupled UQit—a lightweight in-memory library for streaming statistics—with Neko through ADIOS2 [1]’s SST (Scalable, Streaming Transport) engine. At compile time, Neko is instrumented to issue non-

blocking ADIOS2 streaming calls of the flow fields of interest. The SST engine establishes application-to-application channels that stage data in dedicated transport buffers and forward them to any attached reader ranks without performing file IO. Because writers never wait for disk, and SST's handshake protocol overlaps network transfers with computation, the impact on Neko's time-stepping remains negligible.

On the UQit side, we create a dedicated Python sub-communicator via communicator splitting. The UQit ranks then attach as an SST reader to the same in-memory stream. SST delivers each published block as soon as it is available, and because the transfers are all non-blocking and asynchronous, the CFD ranks never pause for UQit ranks. UQit's routines maintain running sums, sums of squares, and in-situ estimates of the autocovariance at each lag. Since everything happens concurrently on the UQ sub-communicator, the solver incurs only the small price of a non-blocking buffer hand-off, and SST's internal progress engine handles all network polling without disturbing the simulation's scalability.

The final step, model fitting of the accumulated autocorrelation and computation of the uncertainty estimator is deferred to a post-processing stage. As the curve-fitting step needs to be performed once and at the end of the simulation, embedding it into the running simulation would force a global synchronisation for a compute-intensive optimisation step causing the Neko ranks to idle. Instead, once SST streams the final timestep, UQit writes out the compressed arrays to the disk. In a separate parallel post-processing step, UQit loads these arrays and performs the least-squares fit to the model Autocorrelation Function (ACF) and writes the HDF5+XDMF output in parallel. This two-phase strategy of non-blocking in-situ accumulation followed by offline model fitting ensures that Neko's performance scales linearly while still delivering uncertainty estimates. The schematic of our workflow is shown in Figure 1. In collaboration with the CEEC project, this approach was applied and tested for the case of a periodic hill and the results are shown in Figure 2.

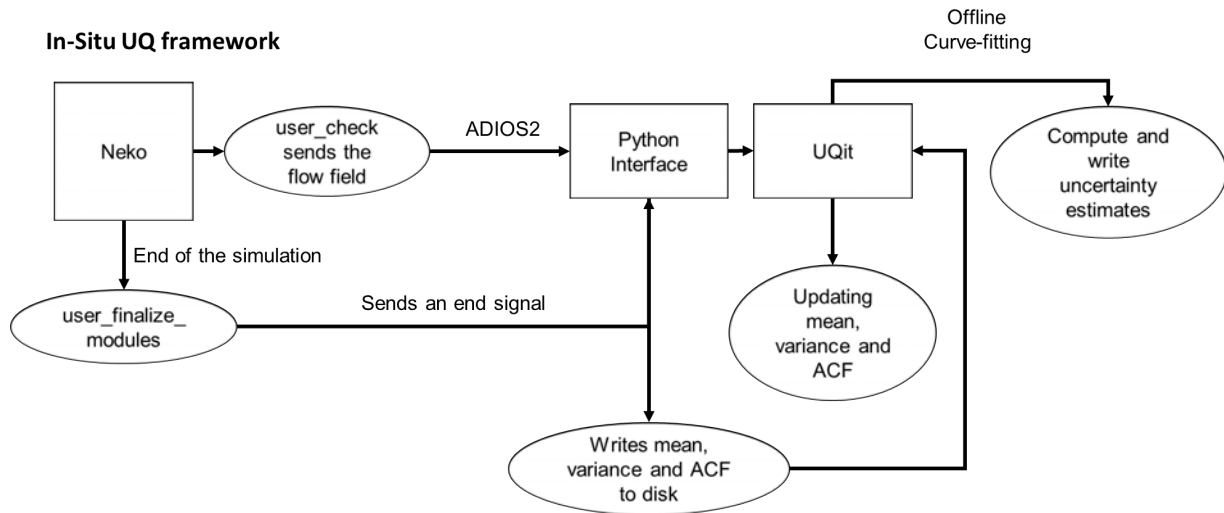


Figure 1: The in-situ workflow for uncertainty quantification between Neko and UQit

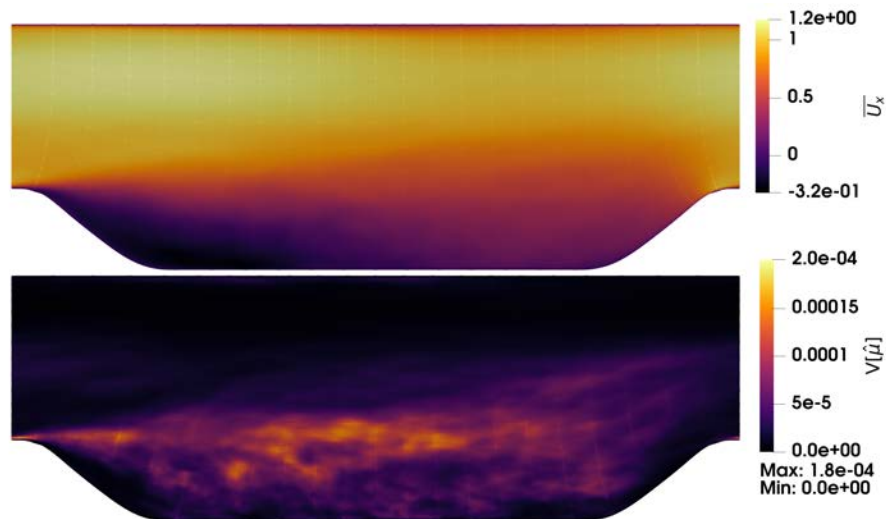


Figure 2: Mean Streamwise component of velocity (top) and the variance of the sample mean estimator averaged over 60 convective time units

A similar in-situ approach has been adopted for our aeroacoustic simulations using a Python aeroacoustic package AcoNeko. In this case, Neko's simulation component non-blockingly streams both geometry (point coordinates, normals, face areas) and dynamic data (pressure and its time derivative) via ADIOS2's SST engine. At startup, each CFD rank publishes a one-time static block of geometry; thereafter, every prescribed time step it pushes the instantaneous pressure and its derivative. On the AcoNeko side, a group of Python ranks, split off from the main communicator, attach as SST readers to this in-memory stream. The static header is read once, masked to the region of interest, and used to build the acoustic receiver grid and its time grid. Subsequent dynamic steps arrive asynchronously and are processed entirely in memory. AcoNeko computes radiation times, interpolation weights, and accumulates the surface-term contributions on the time grid, all overlapping with the flow solution. Once the simulation completes, it writes out the acoustic pressure signals for each observer point.

2.2 L2G, OpenFOAM, Raysect

L2G is a field-line tracing code developed at the University of Ljubljana (UL). The focus has been directed towards the integration of parallel IO capabilities to handle data-intensive operations of storing the intermediate physical information about field lines, such as field-line length, impact angles between the wall and magnetic field-lines, distance of wall from the separatrix, magnetic field and magnetic flux parameters. In principle all these parameters are stored for large number of triangles that define the complex first wall and are further used to calculate heat flux on the wall. Combination of OpenPMD standard and ADIOS2 backend is a solution to overcome the serial IO problem of scalability with increasing Message Passing Interface (MPI) rank counts.

In L2G parallel IO helps to better compress and store the physical parameters to binary-packed formats across MPI processes. Field-line properties are written to disk using the BP4 format with smaller metadata and better latency. In every iteration, the data is flushed to the disc with non-blocking communication between processes. It is expected that this will be the case also for communication between simulation and visualisation of the field lines,

which is also planned within WP4 in the EXCELLERAT P2. This work is expected to progress further in collaborative efforts with Plasma-PEPSC, where in-situ visualisation and high-throughput data exchange is important part of plasma simulation tools.

2.3 SOD2D

SOD2D has been instrumented with a SENSEI interface in EXCELLERAT Phase 1. To make that work SOD2D is linked at compile time to the SENSEI library that in turn links to the analysis backends the user wants to use. This causes a long stack of dependencies which make deployment especially on specialised hardware like HPC machines difficult. Some dependencies were just not ready to be compiled with the NVIDIA HPC compiler, so they had to be compiled with a different compiler than SOD2D. Compile time linkage also limits the supported analysis backends to those that are installed by system administrators if users do not build the SOD2D-SENSEI stack themselves. To solve this, we introduced the CATALYST II [2] interface to SOD2D in Phase 2. CATALYST II uses a lightweight ABI-stable data description model instead of Visualisation Toolkit (VTK) (as with SENSEI). This allows CATALYST II to load user implementations of analysis backends at runtime. This means that SOD2D can be deployed independently of the backends the users want to use, therefore with way less dependencies.

2.4 Vistle

In-situ visualisation is an important part of preparing workflows for exascale, as it removes the very large storage and IO requirements of the traditional pre-processing, solving and post—processing workflow. The amount of data needed for this traditional workflow is no longer an option for exascale, due to the very large amount of data that would be needed, so greatly reducing the storage and IO is a requirement.

During EXCELLERAT P1 the SENSEI [3] in-situ framework was chosen as one of the tools to establish common infrastructure for in-situ capabilities. As part of this an analysis adapter for SENSEI was developed for Vistle [4], and Nek5000 [5] was instrumented with SENSEI. This work is continued in EXCELLERAT P2. Due to changes in the APIs of SENSEI and Vistle the SENSEI – Vistle analysis adapter needed to be updated. The biggest change in this update is that SENSEI’s VTK bridge is now able to couple simulations and analysis tools built with different VTK versions.

The next step was applying the knowledge from instrumenting Nek5000 to Alya [6]. Since both SENSEI and Alya using the CMake [7] build system as build system generator linking it to SENSEI was relatively straight forward. It was also possible to reuse most of the code used in Nek5000 to bridge from the simulations Fortran code to the SENSEI’s C++ interface through C bindings. As an initial step Alya’s grid consisting of high-order Lagrange hexahedron cells is converted to VTK which has built-in support for this type of cells.

The velocity data array is exported to the in-situ visualisation every timestep. The grid does not change during the simulation, and so only needs to be exported once. This is done in the first timestep.

Vistle however does not support high-order cells, therefore, a conversion algorithm was implemented to convert the high-order Lagrange hexahedron cells of order x, y, z used in Alya to the transformed to $x \bullet y \bullet z$ linear hexahedrons needed for Vistle.

The functionality of coupling Alya through SENSEI with Vistle was tested using a small simulation case running on four MPI ranks, and a simple image showing the output of a Taylor-Green vortex simulation (see Figure 3).

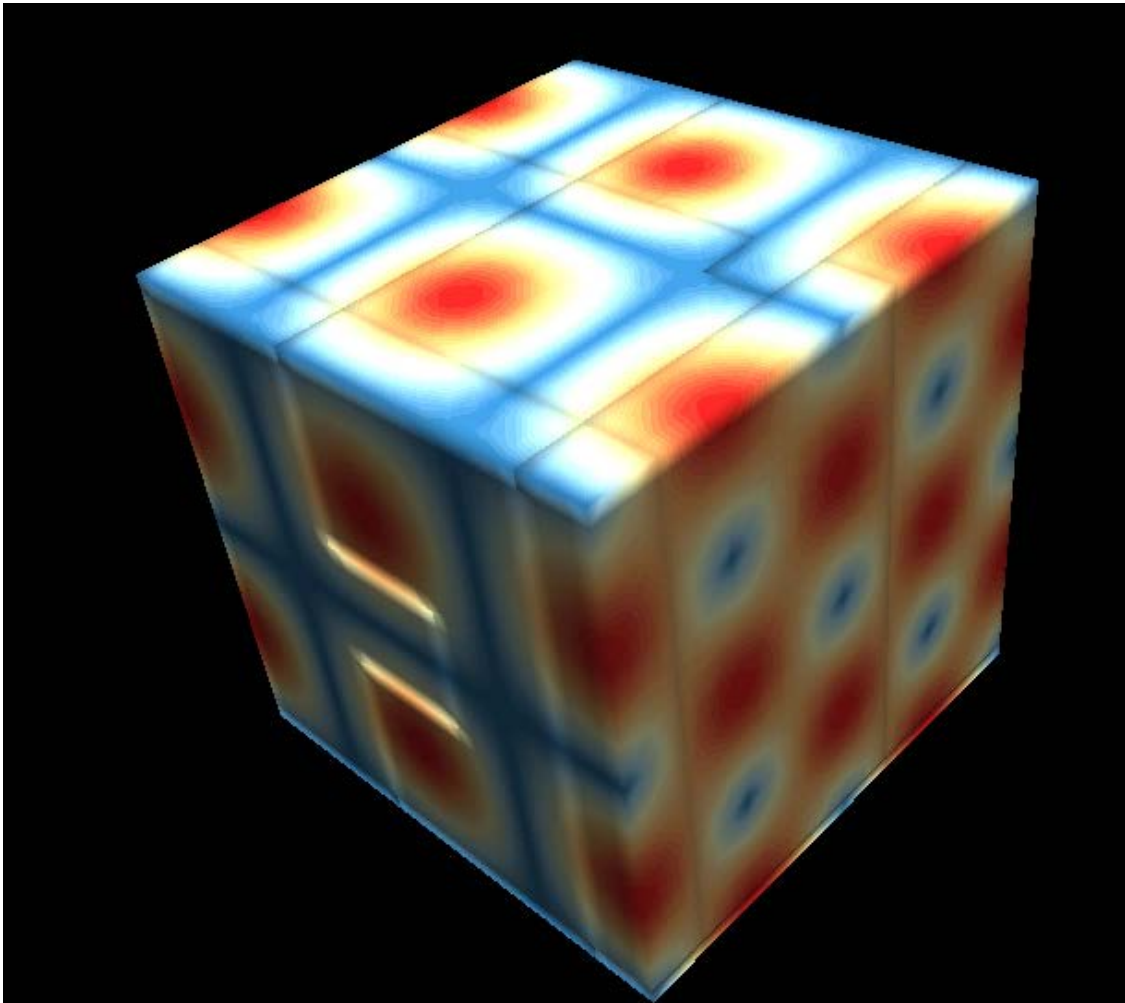


Figure 3: Velocity field of a Taylor Green Vortex obtained using Vistle and the SENSEI interface of Alya

In addition to the in-situ work on Alya, the initial port of the Nek5000 SENSEI implementation has been completed is also underway on adapting Neko [8], with in-situ tools, however technical problems remain before an in-situ visualisation using Neko can be performed.

To match the developments in SOD2D a CATALYST II module has been developed in P2. This module produces a libcatalyst-vistle library that can be configured in CATALYST II instrumented simulations for runtime linkage. As described in the SOD2D section this approach completely decouples deployment of simulations from Vistle as an analysis backend.

2.5 STREAMS

Direct Numerical Simulations at moderately high Reynolds numbers require massive computational grids whose management demands exceptional efforts even at the Input/Output level. In particular, the storage of three-dimensional fields is challenging and must be limited as much as possible. To overcome this difficulty, STREAMS supported in-situ visualisation via ParaView's Catalyst ([2]) technology, which has been successfully used for grids of up to 30

billion points ([9]). In the last 18 months of the project, in-situ support has been substantially improved and expanded, particularly on 4 aspects:

- Porting to Catalyst2: Catalyst2 technology is a complete paradigm shift from Catalyst1. Specifically, it supports Conduit [10] which provides a standard way to describe computational simulation meshes. The implementation within STREAMS has therefore been largely rewritten to support Catalyst2 and is actually much cleaner compared to the previous one. The integration of the in-situ part within STREAMS allows the user to specify the characteristics of the in-situ processing (e.g., passed variables, extraction times) directly from the input file without having to modify the STREAMS source code. The visualisation pipeline is then typically implemented in Python.
- Extension to curvilinear code: since the current version (2.1) of STREAMS incorporates the functionality of the FLEW code, the in-situ implementation has also been extended, especially to support curvilinear grids. Although the structured mesh type is used for both Cartesian and curvilinear grids, some changes have been needed especially to compute derived types to be passed to the visualisation pipeline.
- Backend and deployment: in-situ code is integrated within our `sutils` library for automatic generation of computational backends. In-situ functionality must be enabled during STREAMS compilation and linking with the Catalyst2 API. The configuration is built into the STREAMS Makefile system and is therefore quite straightforward. However, a properly compiled Paraview must be available at runtime, and compiling all components, including their dependencies, is not trivial. We have deployed the entire software stack and successfully tested in-situ execution on five different EuroHPC systems, namely Leonardo-Booster, MareNostrum5-ACC, MeluXina, LUMI-G, and LUMI-C.
- Code publication: the in-situ code is part of the STREAMS-2.1 code ([11]) published open-source on GitHub (<https://github.com/STREAMS-CFD/STREAMS-2>). Contextually, the online reference guide (<https://streams-cfd.github.io/STREAMS-2/>) contains information to facilitate compilation and execution with in-situ. An example case of a turbulent curvilinear channel already configured for use with in-situ is also available.

We plan to test and use in-situ visualisations for challenging simulations which will be performed during the last part of the project.

3 Outer loop optimisation

Shape optimisation is an integral part of design and engineering workflows. It helps in maximizing the performance of structures and minimizing resource wastage while adhering to geometric and design constraints. Previously, engineers had to carry out trial-and-error experiments to determine the optimal shape of physical objects. With the current advancements in the domain of computing, it has become easier to implement shape optimisation in an automated manner without any human interaction.

3.1 Optimisation Workflow

A black-box method called the Bayesian Optimizer (BO) is developed for this purpose and has been tested with an initial use case of NACA airfoil. The shape of the airfoil is constructed using a cubic Bezier curve with seven control points (see Figure 4): Accounting for the x and y co-ordinates, we have 14 parameters, out of which five parameters are fixed, leaving nine free parameters that need to be optimised. We bound the free parameters with box constraints which are provided to the optimizer. These constraints help in eliminate unphysical shapes and ensure that we only obtain smooth and non-intersecting profiles. The goal of the optimisation task is

to minimise an objective value which is defined as $Drag - 0.1*Lift - 0.05*Area$ in the 9-dimensional space.

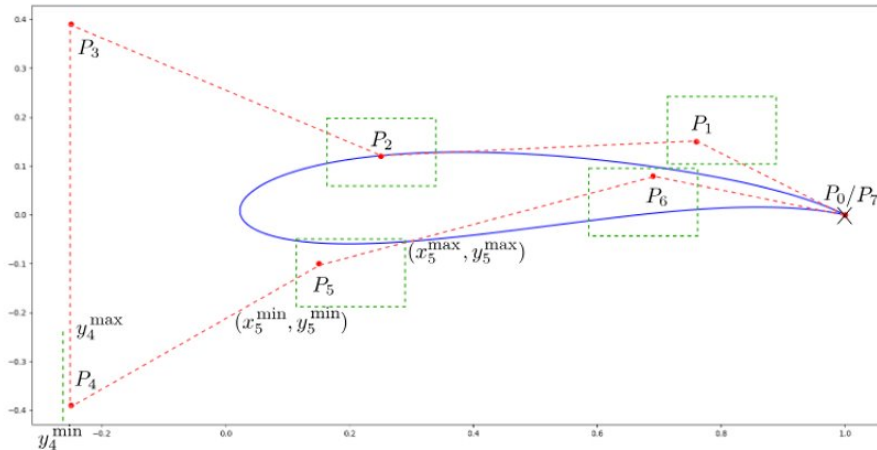


Figure 4: Bezier curve representation of the airfoil with control points

The workflow begins with generating twenty initial seed samples using Latin Hypercube sampling. Here, a sample refers to a 9-dimensional point. An SQL database stores all the information about the samples and a query interface is utilised to fetch and insert the samples into the database. After the initial samples are generated, evaluation of these samples is carried out in parallel. In the evaluation stage, an airfoil profile is created from the sample, followed by meshing, CFD simulation with the m-AIA solver, and computation of lift, drag, and objective value. The BO is called to propose the next sample after the initial samples are evaluated. The samples are generated and evaluated by the BO in a sequential manner until a maximum number of iterations are reached. The entire workflow is managed by a SLURM scheduler that allocates and initiates batch jobs as well as loads the required environments, making the process scalable and efficient.

3.2 Results and Inference

Observations were made on the sample database to identify the best and worst airfoil profiles generated, see Figure 5. The ideal airfoil shape must have a negative objective value that is closest to 0. The best case has a very high lift to drag ratio of 5.8 whereas the worst case only has a value of 1.3. In the best-case scenario, the boundary layer remains well attached over the airfoil with minimal flow separation. A narrow, periodic von Karman vortex street can be observed that indicates steady low-energy losses in the wake. On the other hand, in the worst case, we observe the flow detaching early on the suction side and the formation of separation bubbles. A chaotic wake region that is broad and unsteady is evident which indicates high energy losses and instability.

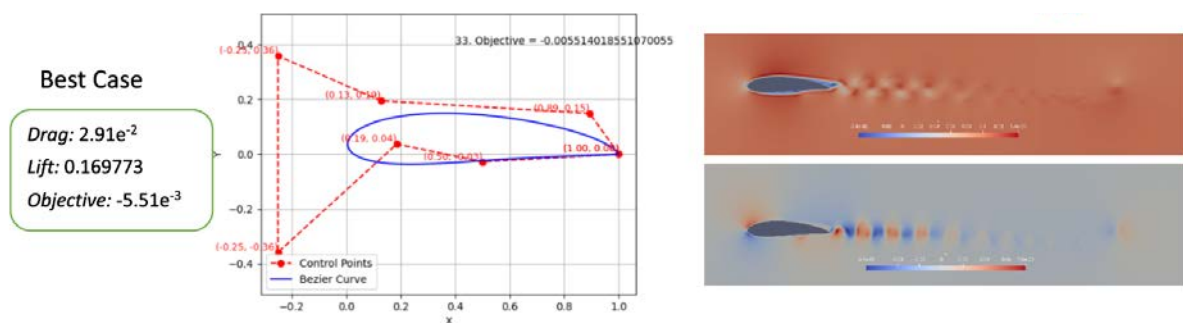


Figure 5: Best airfoil profile generated by the Bayesian Optimizer (left), x-velocity (right, top) and y-velocity (right, bottom) results from CFD simulation of the airfoil

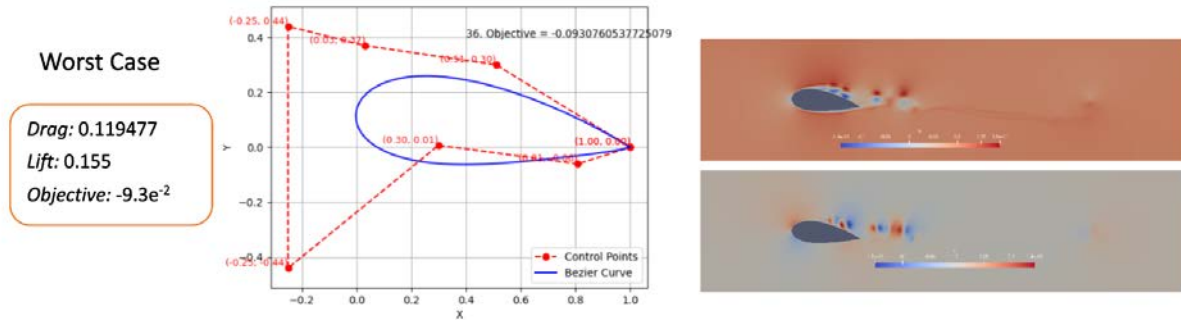


Figure 6: Worst airfoil profile generated by the Bayesian Optimizer (left), x-velocity (right, top) and y-velocity (right, bottom) results from CFD simulation of the airfoil

3.3 Dimensionality Reduction

To understand how the BO chooses the subsequent samples, PCA was used to perform dimensionality reduction on the input data (i.e. the 9-dimensional sample), see Figure 7. We observed that the upper left quadrant shows the most desirable objective values (i.e. points in yellow). Overall, it can be remarked that the BO conducts a very broad search for the optimal points and that it is like prioritizing exploration of the parameter space rather than exploitation. To find a good airfoil profile, it is imperative to tighten the optimizer search around the upper left quadrant.

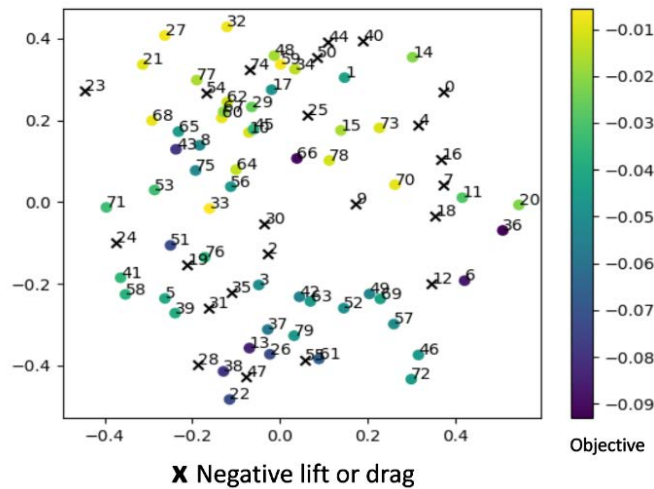


Figure 7: Plot of 2D PCA on input data

In-house tools such as DataViewer and SimExplore are used to perform dimensionality reduction on the output data (simulation results), see Figure 8. Although, no distinct clusters or patterns are distinguishable, a distinct outlier can be identified in these plots.

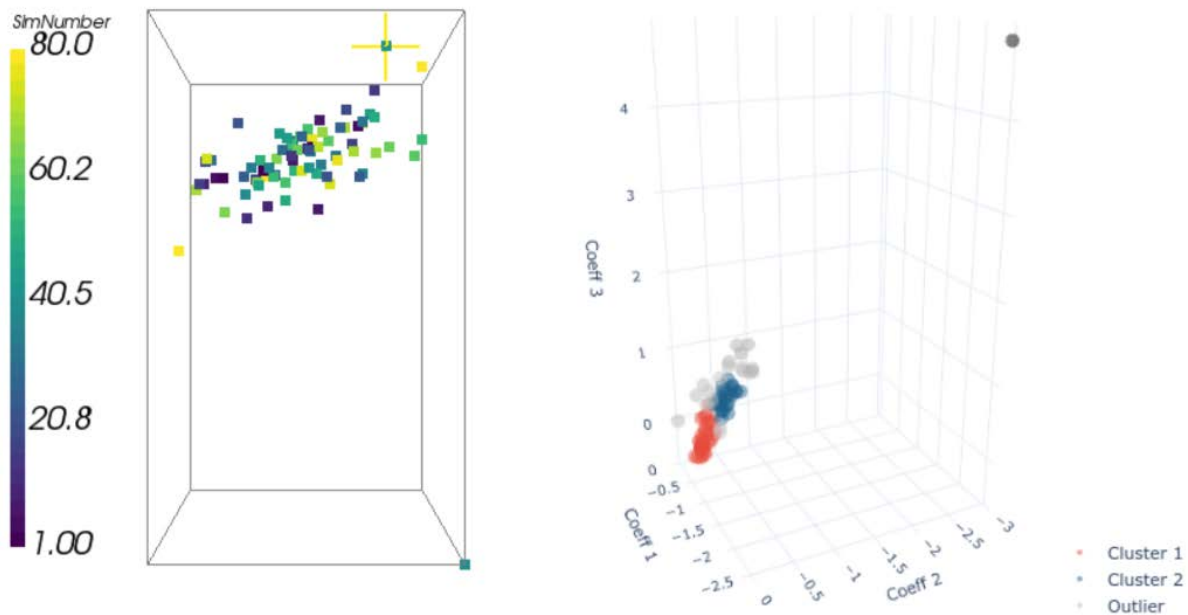


Figure 8: Results from DataViewer (left) and SimExplore (right)

4 Homogenisation/Standardisation of HPC Workflows

Using HPC systems is often complicated, and different HPC systems require different commands and tools to use them effectively. To facilitate the industrial use of CFD applications it is important to create a standardised workflow. A standardised workflow is a significant step in improving the tool maturity, replacing several manual actions with an automated workflow, and simplifying the use of the EXCELLERAT CFD applications as well as reducing training time.

4.1 Tools for Workflow Development

As facilitating the industrial use of CFD through refining the creation of relevant workflows is a key part of EXCELLERAT P2. The initial automated workflows while suitable for research applications, require significant enhancement before they would be suitable for industrial applications. Several tools have been developed to assist this work.

4.1.1 Lemmings

Lemmings is an open-source Python package that is simple to install using python package managers. It was initially developed during EXCELLERAT Phase 1 by CERFACS. This software simplifies the submission of multiple inter-dependent jobs on HPC cluster schedulers. Although originally tailored for CFD applications, Lemmings can be used in many recursive job scenarios. It also incorporates a farming mode that facilitates the replication of recursive jobs for parametric studies. Lemmings offers an efficient solution for automating pre-existing manual workflows while allowing the terminal commands used by conventional HPC to be imported. A typical workflow loop for the Lemmings automation scheme is shown in Figure 9.

The Lemmings LOOP

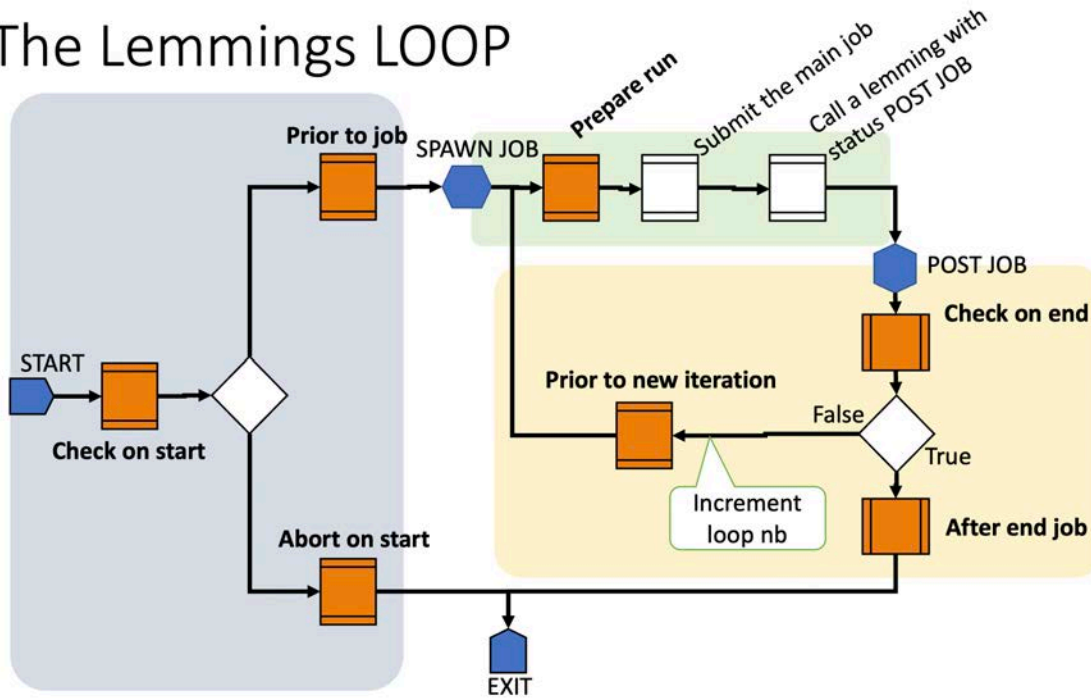


Figure 9: The backbone loop of the Lemmings automation scheme. By injecting custom code into the orange blocks, the user can adapt the tool to most workflows

4.1.2 SCALES

SSC-Services GmbH, in alignment with the European project EXCELLERAT, has introduced the SCALES solution. SCALES was designed to aid SMEs lacking expertise in HPC and integrates workflows into a web portal. The service records each workflow and can subsequently replay it with diverse data feeds. The user experience provided by SCALES resembles that of Gitlab Pipelines, a widely utilised system within the Gitlab community. This approach offers an intuitive learning curve for users, reducing training time and simplifying the use of HPC resources to new users. SCALES powers the Data Management Platform EXCELLERAT service [12]. Figure 10 shows the high-level monitoring dashboard of SCALES.

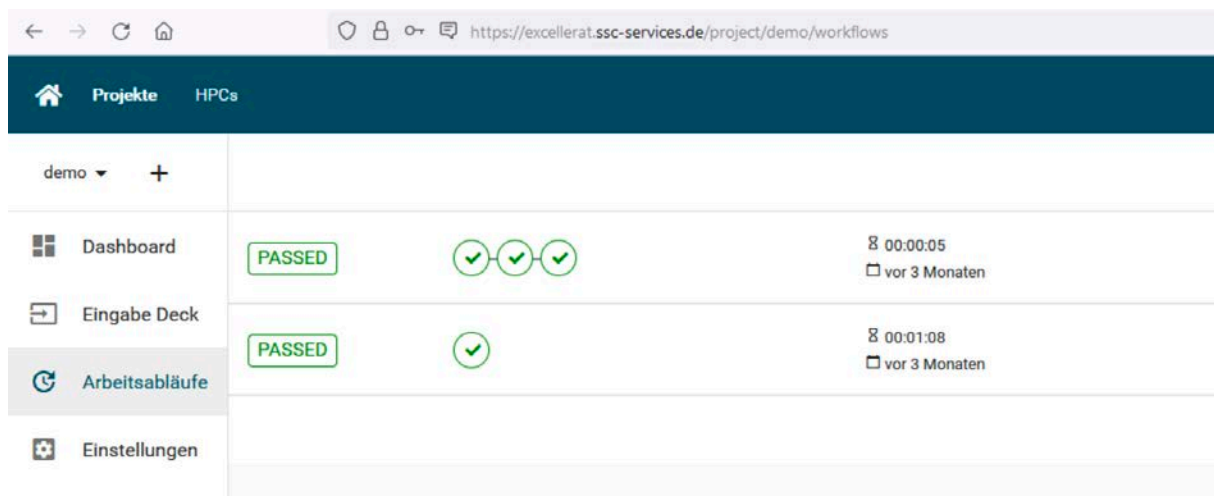


Figure 10: A screen capture of one of the high-level monitoring dashboards of SCALES. Workflows of tasks can be resubmitted on the cluster until satisfactory results are obtained. Unlike the Lemmings approach, all connections and expert commands are encapsulated behind this graphical dashboard

4.2 Recommendations

The initial findings described in the following sections were generated using a test workflow of combustion chamber behaviour analysis provided by the Safran Group [13]. This workflow was analysed using the lightweight open-source tool Lemmings.

While the initial findings were developed using CFD workflows, we believe the insights from studying this workflow are universally applicable across many domains, workflow managers and job schedulers. In the following sections we detail many of these specific insights.

4.2.1 Unified Chronological logging

Most workflows use tools with different and varying origins. These tools encompass research solvers, shell commands, postprocessing utilities, and analysis software, each with its output format and logging mechanisms. These many and diverse outputs can make identification of the source of any issues challenging. Streamlining these diverse log entries into a single, chronological log file proves invaluable in saving time and enhancing collaboration among developers, users, and support teams.

4.2.2 A Unique Identifier for workflow instances

One common challenge faced by new users when working with workflows is managing multiple instances concurrently. Distinguishing between results generated by different instances can become unexpectedly complex, leading to confusion. To address this, we recommend the generation of unique execution IDs, akin to the unique job IDs provided by the scheduler. These IDs serve multiple purposes: they facilitate monitoring and debugging by naming the unified log, enable customisation of job names for clear identification in scheduler queues, and streamline general control commands, simplifying the management of multiple workflow instances.

This idea is implemented in Lemmings by ensuring that each chain of Lemmings jobs on the cluster uses the same name, a user defined prefix followed by two random syllables, followed by two digits. In the following example, this name is `twicer_JEXA93` for all jobs:

```
Status for chain twicer_JEXA93
```

Loop	Solution path	Job end status	progress	CPU time (h)	job ID	pjob ID
0	./	ended, continue	NA	0.001	95575	95578
1	./	ended, continue	NA	0.002	95590	95591
2	Submitted	Submitted	Submitted	Submitted	95611	95612

4.2.3 No File Movement

In our experience, we encountered a common pitfall where archived simulations involved extensive file movement within the workflow. These moves resulted in broken relative paths, rendering archived runs challenging to reproduce. The solution lies in avoiding any file movement within the workflow. All executions should occur within their designated, definitive folders, ensuring a seamless workflow history for easy re-execution.

4.2.4 Version and Document the Workflow

An effective collaboration between developers and customer beta-testers during workflow iterations requires clear versioning. This extends beyond component versioning to encompass the evolution of the workflow itself. Additionally, documenting the workflow strategy within

each job folder, outlining the initial file locations, databases, and mesh, is very important when support is provided by HPC domain experts, who may not be users of the workflow themselves. This approach simplifies future work and enhances communication.

4.2.5 A Sandbox Scheduler

Often HPC facilities offer a “debug” partition for rapid testing of individual runs. Workflows that consist of numerous jobs can still result in significant waiting times. To reduce feedback delays for developers and support teams, we highly recommend adopting a “sandbox scheduler” i.e., a lightweight emulation of a scheduler, as demonstrated by the Lemmings sandbox, which can significantly accelerate testing and debugging of the automated workflows. A sandbox scheduler can also be easily used within a suite of Continuous Integration (CI) tests, simplifying the automatic testing of changes.

The Lemmings sandbox for example supports four commands: *start*, *submit*, *cancel* and *qstat*, i.e. the commands to start the tool, add a new job, cancel a job and show the current queue respectively. Everything is then synchronised by updating a file on disk. The whole sandbox consists of 400 lines of python code and is readable on the Lemmings public repository [14].

4.2.6 A mock-up Solver

Similar to the sandbox scheduler, the concept of a mock-up solver streamlines the testing process. The mock-up solver is a program that takes core HPC solver inputs and rapidly generates dummy outputs, mimicking a real simulation. This testing involving multiple simulations becomes feasible within minutes and considerably enhances the work of the developers and support teams.

In order to create a mock-up solver for the AVBP [15] case, a script was created called “avbp_mockup”. This script reads the main input file, running parameters and the mesh files. It then creates from these the relevant output files, i.e., the solution at specific time points, averaged solutions and monitoring files.

4.2.7 End User Commands

Workflow commands often contain detailed information essential for engineers overseeing the process, yet they can be overwhelming for end users. To strike a balance, we propose segregating “End User Commands” into a separate script, such as a shell script in our case study, managed by the customer. This straightforward approach has proven highly effective, enabling most initial user requests to be addressed directly within the customer’s organisation by refining the user experience.

A typical BASH shell script that could be used for this process is shown below.

```
#!/bin/bash

echo "Loading bash commands for WorkFlow FooBar in your terminal."

# Adapt this to your cluster
export LEM_MACHINE_FILE="mymachine.yml"
export WFLOW_PATH="(…)/WorkFlow_Lemmings/"

#
export LEM_WORKFLOW="${WFLOW_PATH}workflow_FooBar.py"
export LEM_HELPFILE="${WFLOW_PATH}workflow_FooBar.md"
export LEM_INPUT="${WFLOW_PATH}workflow_FooBar.yml"

echo "You will use workflow $LEM_WORKFLOW."
```

```
echo "Machine configuration is at $LEM_MACHINE_FILE."

function lemfoobar_input () {
    echo "Copy workflow FooBar input file in this directory... "
    \cp $LEM_INPUT .
}
echo " - lemfoobar_input to get a default input file"

function lemfoobar_run () {
    echo "lemmings run --inputfile $1 --machine-file $LEM_MACHINE_FILE
$LEM_WORKFLOW "
    lemmings run --inputfile $1 --machine-file $LEM_MACHINE_FILE
$LEM_WORKFLOW
}
echo " - lemfoobar_run myinput.yml to start your FRT workflow"

function lemfoobar_status {
    echo "lemmings status --progress"
    lemmings status --progress
}
echo " - lemfoobar_status to get a status of the last workflow, running or
not."

function lemfoobar_help () {
    cat $LEM_HELPFILE
}
echo " - lemfoobar_help to read about the FRT workflow requirements and
behavior"

function lemfoobar_kill {
    echo "lemmings kill --machine-file $LEM_MACHINE_FILE"
    lemmings kill --machine-file $LEM_MACHINE_FILE
}
echo " - lemfoobar_kill to kill your current workflow."
```

4.2.8 Container issues

Containers, such as Singularity Containers, offer a means to maintain a stable execution environment that can be reproduced across various machines. However, when integrated into workflows, a challenge arises. Containers are typically built on a machine different from that of the workflow creators and lack access to local job submission commands. Addressing this issue requires collaboration with the local IT team to adapt containers, potentially delaying the solution's availability.

4.2.9 Reduced version of workflow recommendations

Initial insights from testing a combustion chamber workflow provided by Safran using the open-source tool Lemmings have led to several recommendations for improving workflow usability and traceability. Key suggestions include unifying log outputs from heterogeneous tools into a single chronological log file, assigning unique identifiers to each workflow instance for easier monitoring and debugging, and avoiding file movement during execution to preserve reproducibility. Additionally, versioning workflows and documenting job configurations are essential for effective collaboration between developers and HPC support teams. End-user experience can also be improved by providing simplified command wrappers tailored to their needs.

To streamline testing and support, the use of mock-up solvers and a lightweight "sandbox scheduler" is encouraged. These tools simulate core components of the workflow environment—such as job queues and solver behaviour—allowing rapid iteration without waiting for actual resource allocations. The sandbox and mock-up strategies have proven especially useful in Continuous Integration (CI) pipelines. Finally, while containers like Singularity provide reproducible environments, they may lack compatibility with HPC-specific job submission systems, requiring adjustments by local IT teams to ensure smooth integration within production workflows.

4.3 Automatic static mesh refinement

Meshing is an important first step in CFD, where the complex object to be modelled is converted into a mesh of well-defined cells where the governing physical equations can be applied, allowing the solver to simulate the physical behaviour. The traditional method of generating a mesh involves initially generating a final-sized mesh, then refining it iteratively by running simulations using that final-sized mesh. Once the mesh has been sufficiently refined it can be then used in production simulations.

Unfortunately, this method is no longer viable at the exascale level. Using many runs at the final mesh size is too expensive. Therefore, a different method is required. In order to generate a mesh suitable for exascale at an appropriate cost, it is necessary to use an initial mesh that is significantly smaller than the final-sized mesh desired, and iteratively increase the mesh size as more information from simulations are obtained. For each mesh size a simulation will give more information on where the mesh should be refined. When successful this process would provide sufficiently good mesh for the final simulations at a controlled cost.

Note, unlike Dynamic Mesh Adaptation, this process discards the simulation history, and retains only the final mesh and solution. Indeed, there are theoretical open issues with high fidelity simulations obtained on a moving mesh, so to stay on the safe side this workflow is an Automated Static Mesh Refinement (ASMR).

The initial goal of this work is to show that it is possible to replace the domain knowledge needed to create a mesh of sufficient quality with a mechanical iterative process. A secondary goal is to demonstrate the mesh generation procedure that can reliably produce a stable grid with more than one billion cells. It is also important to understand the most efficient way to produce the grids, i.e., is it better to use many small mesh refinements or aim for fewer but larger refinements?

Initial runs using ASMR, the CFD code AVBP have been completed using EXCELLERAT use case 2 (hydrogen combustion for propulsion). The ASMR runs have to date produced acceptable meshes for industrial configurations. To date meshes of up to 50 million cells have been generated using this method, and these meshes are currently being reviewed to ensure that they are high enough quality.

In order to hit the target of generating a mesh large enough for exascale applications i.e., of around a billion points, the workflow must be adapted to use parallel mesh development. An example of ASMR is shown in Figure 11, where the initial mesh of 2.8 million cells is in several stages into a mesh of 46 million cells.

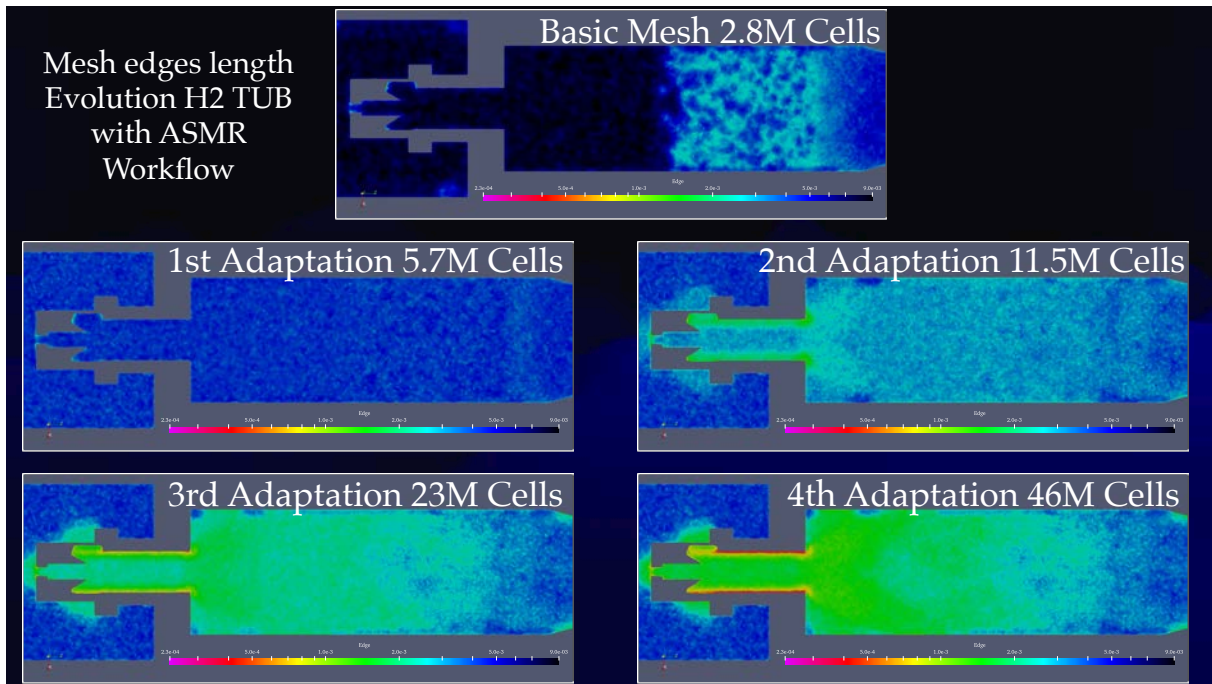


Figure 11: Figure showing how a mesh is automatically refined using mesh adaptation with the ASMR workflow. The configuration is an academic Hydrogen Burner (TUB burner). Our final goal is the generation of a 1 billion cell mesh in a single workflow

4.4 Monitoring Applications with Job Data Points

Several EXCELLERAT P2 use case prototypes are already in the hands of their final users—for example, engineers at Safran are actively working with Use Case 2. The next challenge is to monitor the performance of these workflows in real production environments, despite changes in software versions, workflow configurations, or the computational resources used. For instance, when users switch to a GPU-enabled partition, the time-to-solution may vary significantly. However, with each change—be it in resources or user behaviour—many performance-affecting parameters can also shift, such as job duration, compiler settings, or I/O frequency.

To better understand performance in real conditions, EXCELLERAT P2 partners are now focusing on the systematic collection of *Job Data Points*. Initial tests are currently gathering four key types of information:

1. *Input Parameters*: Since software varies, each use case requires a tailored conversion of input data into a JSON-friendly format. This can include information like the degrees of freedom (related to mesh parameters), models used, and how frequently results are stored and how much data is generated.
2. *Job Information*: Extracted from the job scheduler (e.g., sacct for SLURM), this includes metadata such as the user name, job duration, number of processes and partitions, and peak memory usage.
3. *Execution Parameters*: Many HPC applications include internal performance monitoring. Again, custom formatting is needed. Tracked metrics might include total computation time, time step durations, partitioning time, iteration time, software version, and compilation options.
4. *Machine Information*: Tools like *MachineState* provide introspection into the computational environment. This includes hardware and software details such as the processor model, loaded modules, and system-level software versions.

A script runs automatically at the end of each job to collect this information into a JSON file, stored in the user's workspace. Users retain control over whether and how they share this data.

Currently, this approach is being tested on the workflows of Use Case 2, with the goal of generating meaningful insights from real production runs. This methodology contrasts with traditional performance benchmarking, where production features are often disabled to maintain reproducibility. If the insights gained prove valuable, the systematic collection of Job Data Points could be extended to other EXCELLERAT P2 use cases.

5 Since D4.1: Progress Summary and Key Metrics

Table 1 summarises key changes and improvements since Deliverable D4.1, with a focus on quantifiable KPIs where applicable.

Area	Progress Since D4.1	KPI (if available)
In-situ visualisation (Neko, UQit)	Integration of non-blocking streaming using ADIOS2 SST between Neko and UQit for in-situ statistics.	IO overhead: ~0% (non-blocking)
In-situ visualisation (STREAmS)	Migration to Catalyst2 with support for curvilinear grids; tested on 5 EuroHPC systems.	Grid size supported: up to 30B points
Outer loop optimisation (Airfoil)	Bayesian optimiser implemented for shape optimisation with sequential sampling via SLURM.	Best lift-to-drag ratio: 5.8
Automated mesh refinement (ASMR)	Iterative mesh growth up to 50M cells using AVBP for hydrogen burner case.	Current mesh size: 50M cells
Workflow tools (Lemmings, SCALES)	SCALES dashboards introduced; Lemmings supports unified job tracking with ID system.	
Job data monitoring	Initial implementation of automatic job metadata collection via end-of-job scripts.	Coverage: Use Case 2 only
File IO strategy	Expanded use of ADIOS2 and Catalyst2 to reduce intermediate file generation and storage load.	
Workflow reproducibility and CI	Use of sandbox scheduler and mock solvers for faster testing and workflow validation.	CI-ready workflows: partial

Table 1: Summary of key changes and improvements since Deliverable D4.1

6 Conclusions

The EXCELLERAT P2 project has made clear progress toward enabling scalable, efficient workflows for HPC-based engineering simulations. In-situ visualisation has been successfully demonstrated with tools like Vistle and Alya, and the initial integration with Neko and acoustic solvers has shown promising results.

Outer loop optimisation workflows using Bayesian methods have been tested with success on standard airfoil cases. In parallel, the introduction of workflow tools such as Lemmings and SCALES has supported better usability and automation, helping reduce complexity for industrial partners.

The ASMR pipeline has generated high-quality meshes up to 50 million cells, and work continues to scale this to the billion-cell level. Finally, job data point collection from real runs marks a step toward more robust performance monitoring and optimisation in production environments.

7 References

- [1] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, *et al.*, "Adios 2: The adaptable input output system. A framework for high-performance data management," *SoftwareX*, vol. 12, p. 100561, 2020.
- [2] A. C. Bauer, B. Geveci, and W. Schroeder, *The ParaView Catalyst User's Guide v1.0*, Kitware, 2013.
- [3] SENSEI. [Online]. Available: <https://sensei-insitu.org/>
- [4] Vistle. [Online]. Available: <https://vistle.io/>
- [5] Nek5000. [Online]. Available: <https://nek5000.mcs.anl.gov/>
- [6] Alya. [Online]. Available: <https://www.bsc.es/research-development/research-areas/engineering-simulations/alya-high-performance-computational>
- [7] CMAKE. [Online]. Available: <https://cmake.org/>
- [8] Neko. [Online]. Available: <https://neko.cfd>
- [9] F. Salvatore, A. Memmolo, D. Modesti, G. Della Posta, and M. Bernardini, "Direct numerical simulation of a microramp in a high-Reynolds number supersonic turbulent boundary layer," *Phys. Rev. Fluids*, vol. 8, no. 11, 2023. [Online]. Available: <https://doi.org/10.1103/physrevfluids.8.110508>
- [10] Conduit, "A successful strategy for describing and sharing data in situ," presented at the ISAV Workshop, SC 22, Dallas, TX, Nov. 13, 2022.
- [11] F. Salvatore et al., "STREAMS-2.1: Supersonic turbulent accelerated Navier-Stokes solver version 2.1," *Comput. Phys. Commun.*, Art. no. 109652, 2025.
- [12] Data Management Platform. [Online]. Available: <https://services.excellerat.eu/viewcode/9>
- [13] Safran Group. [Online]. Available: <https://www.safran-group.com/>
- [14] Lemmings Public Repository. [Online]. Available: <https://gitlab.com/cerfacs/lemmings/tree/master/src/lemmings/sandbox>
- [15] AVBP. [Online]. Available: <https://www.cerfacs.fr/avbp7x/>