D2.3: Final report on multiscale UQP and V&V

Due Date	14 December 2021
Delivery	14 December 2021
Submission of updated version	N/A
Lead Partner	UvA
Dissemination Level	Public
Status	Final
Approved	Executive Board
Version	V1.7

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 800925.

DOCUMENT INFO

Date and version number	Author	Comments
21.07.2021 v1.1	V. Krzhizhanovskaya	Skeleton
24.09.2021 v1.2	E. Raffin, P. Karlshoefer	Addition of main text
5.10.2021 v1.3	D. Ye	Addition of main text
8.10.2021 v1.4	D. Groen, D. Suleimenova	Addition of main text
13.10.2021 v1.5	W. Edeling	Addition of main text
25.10.2021 v1.5	D. Coster	Addition of main text
26.10.2021 v1.6	V. Krzhizhanovskaya	Editing
28.10.2021 v1.6	W. Edeling	Internal review
12.11.2021 v1.6	E. Raffin	Internal review
15.11.2021 v1.7	V. Krzhizhanovskaya, D. Ye	Final editing

CONTRIBUTORS

1. Valeria Krzhizhanovskaya (UvA) – Editor, Author, Reviewer
2. Wouter Edeling (CWI) – Editor, Author, Reviewer
3. Dongwei Ye (UvA) – Author
4. Diana Suleimenova (UBRU) – Editor, Author
5. Derek Groen (UBRU) – Author
6. David Coster (MPG) – Editor, Author
7. Dongwei Ye (UvA) – Author
8. Paul Karlshoefer (BULL) – Author
9. Erwan Raffin (BULL) – Author, Reviewer

Disclaimer

TABLE OF CONTENTS

1. Executive summary

In this deliverable we cover the works on task 2.3 (*Advanced multiscale UQ algorithms, including intrusive approaches*), task 2.4 (*Advanced UQP development based on task 2.3*), task 2.5 (*Multiscale Verification & Validation*) and task 2.6 (*Scalable UQPs for current petascale and emerging exascale*) in Work Package 2 of the VECMA project.

More specifically, we present advanced multiscale uncertainty quantification (UQ) algorithms including multiple surrogate modeling techniques for multiscale simulations and dimension reductions methods for UQ inputs space. A final consolidated description of Uncertainty Quantification Patterns (UQPs) and Verification & Validation Patterns (VVP) for multiscale simulations, and corresponding implementations in the VECMAtk are also presented. We discuss the scalability of UQPs on current petascale and emerging exascale machines.

2. UQ algorithms and VV patterns

2.1 Introduction

VECMA project Work Package 2, entitled *Algorithms & Formalisms*, is focused on designing and formalizing algorithms for multiscale VVUQ (Verification, Validation and Uncertainty Quantification). In this report, we present the final results obtained in Work Package 2 of the VECMA project: a final description of advanced multiscale uncertainty quantification algorithm, Uncertainty Quantification Patterns (UQPs) for multiscale Uncertainty Quantification (UQ), Verification & Validation Patterns (VVP) for multiscale simulations and corresponding implementation in the VECMAtk. Their scalability as we move toward the exascale is also presented. We discuss in detail the results from task 2.3 (*Advanced multiscale UQ algorithms, including intrusive approaches*) task 2.4 (*Advanced UQP development based on task 2.3*), task 2.5 (*Multiscale V&V*) and task 2.6 (*Scalable UQPs for current petascale and emerging exascale*).

2.2 Advanced Multiscale UQ Algorithms

In deliverable D2.2 “Report on advanced multiscale UQ algorithms, including intrusive approaches, and mapping thereof in UQPs and first results on V&V”, we divided the advanced multiscale UQ algorithms into two parts, namely surrogate modeling and dimension-reduction for multiscale

models. Here we will utilize the same breakdown and describe the new techniques we implemented, on top of those already described in D2.2.

2.2.1 Surrogate modeling

Very briefly, a surrogate model is essentially a cheap approximation of an (expensive) computer code, trained on a number of evaluations of the original model. Once trained, the surrogate takes the place of the expensive code in order to enable tasks that require many evaluations, such as forward uncertainty propagation, sensitivity analysis, inverse problems etc. Here, our particular focus lies on the creation of surrogates for the *small (micro) scales* of a multiscale problem, as these account for the lionshare of the computational burden, see Figure 1.

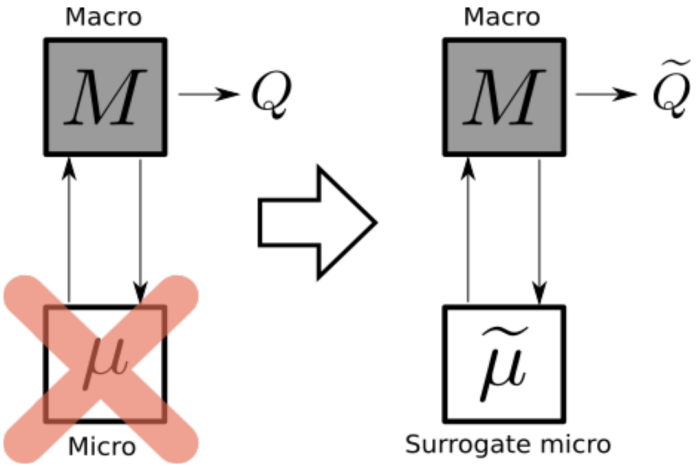

Figure 1: Schematic of a multiscale model consisting of a macromodel M coupled to a micromodel μ. The QoI (Quantity of Interest), denoted Q, is an output of the macromodel. The computationally most expensive component (often the micromodel) can be replaced by a surrogate.

Figure 1 displays an example of Uncertainty Quantification Pattern 3 (described later in D2.3.3), in which there is a cyclic coupling between the large (macro) model and a (single) small-scale model. In D2.2, we already described several techniques for computing surrogate models for the small-scales, namely stochastic surrogates, reduced surrogates and surrogates based on convolution neural nets. Here, we will not repeat ourselves and instead focus on the *coupling between the physical (large-scale) model and the data-driven (small-scale) surrogate.*

Coupling data-driven surrogates and physical (PDE-based) models

Ideally, we could train a small-scale surrogate model on available data, replace the original small-scale model and integrate the coupled system in time, for instance with the goal of obtaining accurate

time-averaged statistics. We have shown that this can work, see for instance our work in [1], which applied our stochastic surrogates to a simplified atmosphere model. However, there is *no general guarantee for stability* in coupled PDE-surrogate systems. Consider for instance the results of Figure 2, which shows an application of our reduced surrogates, involving a neural network component, applied to the 2D Navier Stokes equations. The left subplot shows a snapshot of the 2D large-scale vorticity. More importantly, the right subplot shows the time evolution of a quantity of interest (QoI), for both the large-scale PDE + reference small-scale model (noted Z^{ref}), and the large-scale PDE + surrogate small-scale model (noted Z). The reference small-scale model was extracted from a high-resolution reference simulation. The two QoI curves overlap in the beginning, but diverge at some point in time. This is to be expected due to the chaotic nature of the system. However, the coupled PDE + surrogate system clearly becomes unphysical later on, as seen due to the unrealistic sharp drop in the QoI.

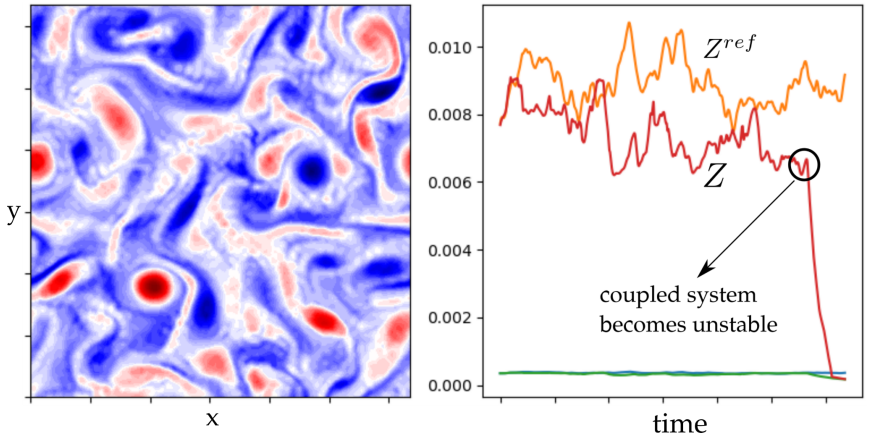

Figure 2: Left: the vorticity contours of the 2D Navier Stokes equations at a single point in time. Right: the time evolution of the enstrophy (Z), for both the large-scale PDE with a reference small-scale model (orange line), and the same large-scale model with a reduced surrogate for the small scales.

As mentioned, the reduced surrogate incorporates a neural network, which is trained *offline*, i.e. on data alone. However, in a UQP3 setting, the surrogate will be subjected to a two-way coupling with the large-scale model (see Figure 1). Hence there is a disconnect between what the surrogate is trained for (prediction of small-scale *data*), and what it is ultimately meant for (performing well in a two-way coupled PDE-surrogate *modeling* environment). Small errors can accumulate over time, leading to the failure shown in Figure 2. This is not limited to our reduced surrogates: other authors have reported similar issues with different surrogates [2].

Hence, for the UQ patterns we describe here, the disconnect between a purely data-based training phase and a predictive phase that includes interaction with physical (large-scale) models needs to be addressed. We are currently investigating *online* training methods, which basically involve a second

training phase while the surrogate is actually coupled to the large-scale PDE. In particular, we are currently adapting the methodology of [2] to our reduced surrogate techniques, described in [3]. Briefly, one of the main challenges is that in an online setting, data points to train the surrogate arrive one-at-a-time, since the large-scale model is advanced one time step at a time. This is very different from offline training, where an entire database of training data is available *a priori*. Whether this 'trickle' of online data points is always sufficient to adapt the surrogate quickly enough is an open question, especially if the surrogate contains a large machine-learning (ML) model. The reason we focus on our reduced surrogates is that for these, the amount of required training data is naturally compressed (hence the name 'reduced', see D2.2 for more explanations). Some initial (yet unpublished) results are shown in Figure 3. Similar to Figure 2, it shows the time evolution of QoIs for the 2D Navier Stokes case, for both the large-scale PDE + reference small-scale model, and the large-scale PDE + surrogate systems. However, this time the latter system is subject to (continual) online training. Now, the reference QoI values and computed values are indistinguishable from each other. Hence, even though the surrogate is coupled to the large-scale PDE, the training phase performs *as if the surrogate was trained offline on data alone.*

Online training is expensive, as it involves running a reference (high-resolution) model in the background to generate the online data points. Next steps therefore will involve assessing the performance *after* the online training phase, and investigating the use of occasional, intermittent online-learning phases.

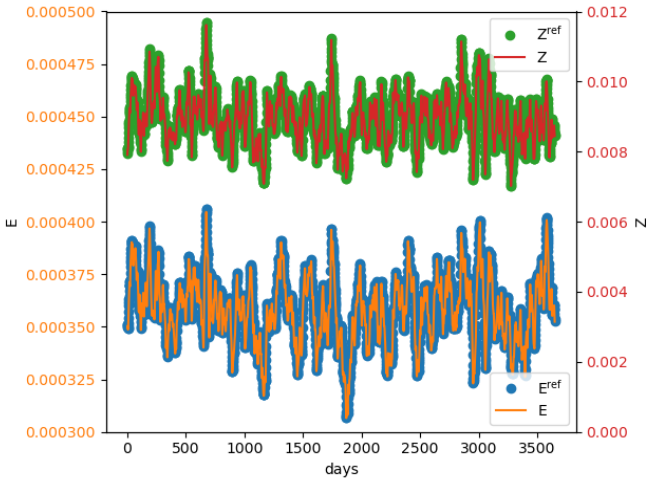

Figure 3: The time evolution of the enstrophy (Z) and energy (E), extracted from the large-scale vorticity fields of the 2D Navier-Stokes equations. The dots represent the values of the large-scale PDE with a reference small-scale model, and the solid lines indicate the solution extracted from the large-scale PDE with a reduced small-scale surrogate, subject to online learning.

2.2.2 Dimension reduction

Here we will focus on newly implemented techniques for dimension reduction of (single-scale) models. These will most often be used in a UQP1/2 setting, where the input uncertainty is propagated through an expensive (multi-scale) model, see the figure below.

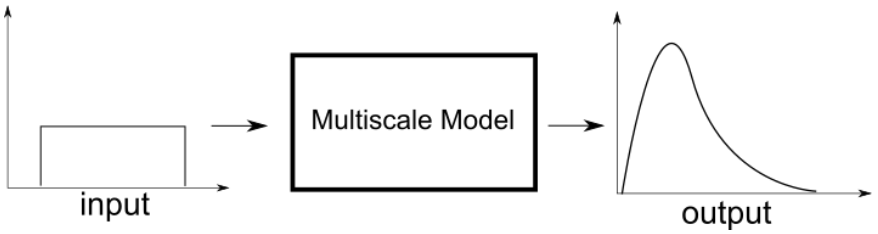

Figure 4: a sketch of the forward uncertainty propagation problem (UQP 1). A probability density function is assumed at the input, and the task at hand is to discern the corresponding output distribution from it, by propagating random input samples through the (multiscale) model.

However, in multiscale models the dimension of the input space can be very large, increasing the cost of forward uncertainty propagation. It therefore makes sense to decrease the dimension of the input space if possible. D2.2 describes the use of sensitivity analysis in this context. Here, we will discuss dimension-adaptive stochastic collocation and deep-active subspace techniques. These are examples of Uncertainty Quantification Pattern 1-A, described later in section 2.3.1.

Dimension-adaptive stochastic collocation

Traditional Stochastic Collocation (SC) can show exponential convergence, depending upon the regularity of the function, and the number of uncertain inputs $\boldsymbol{\xi} = [\xi_1, \cdots, \xi_D]$. At the heart of the SC method lie 1D quadrature rules. The corresponding quadrature points are used to create a sampling plan in D dimensions via simple tensor product, see Figure 5a for an illustration. However, if the number of inputs (D) is more than (typically) 5 or 6, the exponential growth of the tensor product construction quickly yields an inordinate amount of samples that must be evaluated.

That said, even if a model has D > 6, the influence of each parameter on the output will not be equal. In practice, we often observed that only a handful of inputs are really influential. The key to dimension reduction is therefore the identification of these influential parameters. Dimension-adaptive methods [4] iteratively refine the sampling plan based on a suitably chosen error metric, see Figure 5b. Usually, one starts with a single sample in the first iteration, and evaluates the model in various 'candidate' directions. In the case of the SC method these correspond to different candidate 1D quadrature rules of higher order, the 'x' symbols of Figure 5b. For each of these

locations an error/sensitivity metric is computed, for instance, the observed change in the output variance. Only the direction with the largest error is added to the sampling plan, and the cycle repeats. In this manner, an anisotropic sampling plan is created in steps, in which uninfluential parameters do not get refined.

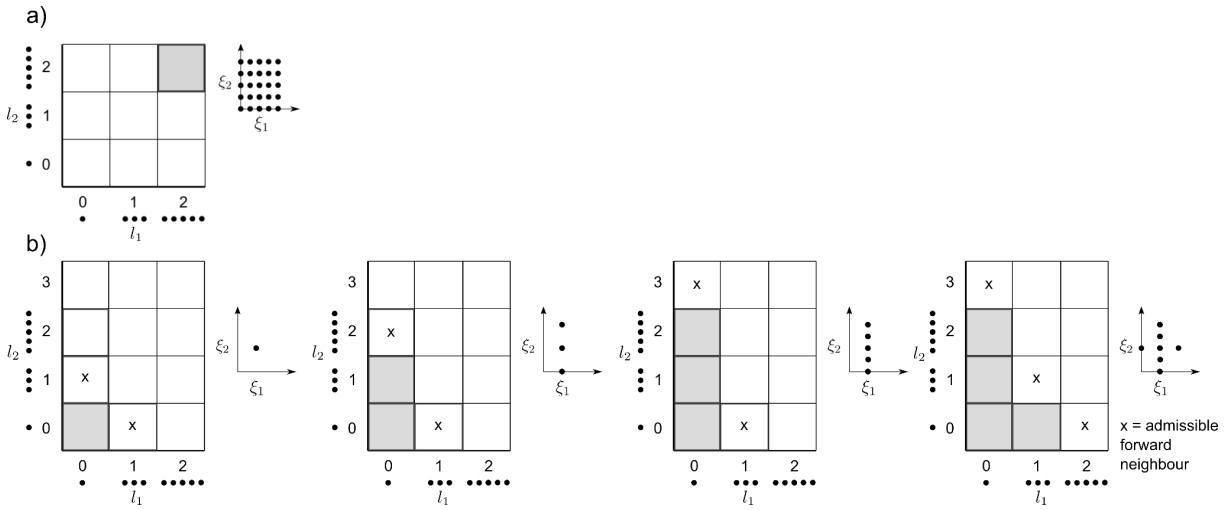

Figure 5: a) construction of standard SC sampling plan in 2D via a tensor product of 1D quadrature rules. b) iterative construction of a dimension-adaptive SC sampling plan in 2D. The starting point is a single sampling point, and additional quadrature rules are added based on their perceived importance.

We applied this method to a well-known British covidmodel [5], where we focused on amplification of uncertainty from the input to the output, and to a molecular dynamics code [6], where we investigated the influence of the random number generator. A similar study, using the EasyVVUQ MC sampler, was performed on a Dutch Covid model, see [7]. These applications will be discussed in more detail in the upcoming deliverable D4.4 “Report on the implementation of non-intrusive and intrusive VVUQ techniques”. Here, we note that both models required the use of HPC resources. The covid model ensembles were executed at the PSNC Eagle supercomputer and the MD ensembles were run at the LRZ SuperMUC-NG machine. A single forward propagation of uncertainty for the covid model required of the order of 10K core hours, and we performed various such studies. We performed a single MD study that required 2M core hours, which was the *a-priori* budget that was available for this study.

As such, the interface between EasyVVUQ (where dimension-adaptive SC is implemented) and the various VECMA HPC tools (FabSim3, QCG-PilotJob) were vital to manage the workflow and HPC job submission. Without these tools, it would not have been possible to realize these studies, and apply uncertainty quantification at scale.

Deep-active subspaces

A downside of dimension-adaptive SC is that it can take many iterations to converge, as many candidate directions will only add a few new sample points, thus leading to small ensembles. At the other extreme of this spectrum are methods that only require a single (large) ensemble, and which work out which inputs are influential in a post-processing step. Active subspaces belong to this category.

The Active Subspace method, introduced by [8], is a class of methods for forward propagation of uncertainty in high-dimensional input spaces. It projects the input vector to a lower-dimensional subspace $\mathbf{y} \in \mathbb{R}^d$, via a tall-and-skinny matrix $W_1 \in \mathbb{R}^{D \times d}$ of orthogonal basis vectors. The active subspace is thus given by $\mathbf{y} = W_1^T \boldsymbol{\xi} \in \mathbb{R}^d$.

Because $\mathbf{y}$ is a linear transformation of the inputs $\boldsymbol{\xi}$, it opens up the possibility of finding directions along which the model varies most. This is especially useful if a model varies significantly in a direction that is not aligned with the coordinate axes of $\boldsymbol{\xi}$, see Figure 6 below for a 2D example.

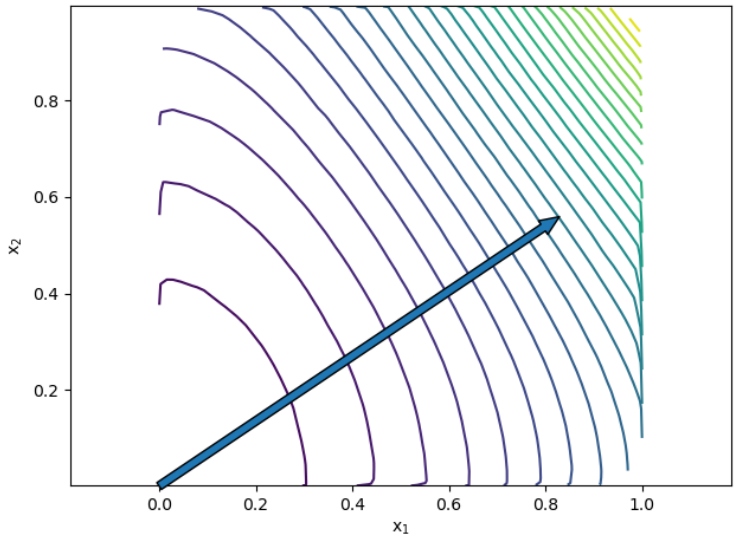

Figure 6: A 2D parameter space (x1,x2) with a 1D active subspace (blue arrow)

In "classical" active subspaces, dimension reduction is achieved by rotating the coordinate system such that it is aligned with the directions of most variability, after which only the most dominant directions are retained. To find these directions, the following average gradient matrix is constructed:

$$C = \int \left(\nabla f\left(\boldsymbol{\xi}\right)\right)\left(\nabla f\left(\boldsymbol{\xi}\right)\right)^T p(\boldsymbol{\xi}) d\mathbf{x}$$

Here, $p(\boldsymbol{\xi})$is the chosen probability density function (pdf) of the inputs $\boldsymbol{\xi}$. Since C is a symmetric, positive semi-definite matrix, it has the following spectral decomposition

$$C = [W_1 W_2] \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} [W_1 W_2]^T$$

Hence the matrix W_1, used to project $\boldsymbol{\xi}$ to $\mathbf{y}$, are the first d eigenvectors of C, which in turn correspond to the d largest eigenvalues in Λ_1. The matrix C can be approximated using Monte Carlo, and this is where the aforementioned large ensemble enters. However, note that we require samples of the *derivatives* of f, rather than samples from f itself.

Derivative information is not always available for computational models. We therefore implemented so-called Deep-Active Subspaces [9], in which artificial neural networks (ANNs) are used to find W_1 via the back-propagation algorithm. Like the classical active subspace method, the construction $\mathbf{y} = W_1^T \boldsymbol{\xi}$ is retained. Moreover, the column vectors of W_1 still form an orthogonal basis. The difference is that the column vectors of W_1 are no longer the eigenvectors of the gradient matrix C, but instead are constructed using Gram-Schmidt orthogonalization. The use of ANNs is motivated by the fact that the active-subspace projection $\mathbf{y} = \Phi\left(W_1^T \boldsymbol{\xi}\right) = W_1^T \boldsymbol{\xi}$ also occurs in a layer of a neural network if the activation function $\Phi(\cdot)$ is linear. Thus, we can interpret the matrix W_1 as a weight matrix of the first hidden layer (with d neurons and linear activation), connected to an input layer through which $\boldsymbol{\xi}$ is passed, see Figure 6.

We implemented the Deep-Active Subspace method from [9] in EasySurrogate, and are currently working on some theoretical extensions of the method. Compared to [9], we derived an analytical form for the derivative of Gram-Schmidt vectors (which is required for back-propagation), and are working on a proof that the W_1 matrix found through back-propagation spans the same subspace as the dominant eigenvectors of C (i.e. the 'classical' W_1 defined above), if C is constructed using the gradient of the ANN. We are also using the gradient of the ANN (which can be computed using back-propagation) to compute global, derivative-based sensitivity indices, which we can contrast with the (variance-based) Sobol indices computed via EasyVVUQ.

Finally, we applied this method to the same British Covid model as in [5], using 1500 samples generated in a single ensemble on the PSNC eagle machine. Here we use the same 19 uncertain inputs as in [5], and the derivative-based sensitivity indices from the deep-active subspace network

highlight exactly the same subset of input parameters as influential, when compared to the Sobol indices we computed via dimension-adaptive SC in [5]. In further studies, we have increased the number of inputs to 41 without issue, and are now working on a 50 dimensional case. So, ML-based surrogates as presented here scale very well to high input dimensions. Although with ML approaches one has to be wary of the well-known generalization error, our initial results indicate that they might already be useful for eliminating uninfluential parameters from a large input set. If the remaining set of influential parameters is small enough, one could also use more traditional surrogate modeling methods to construct a surrogate on this reduced set of important inputs.

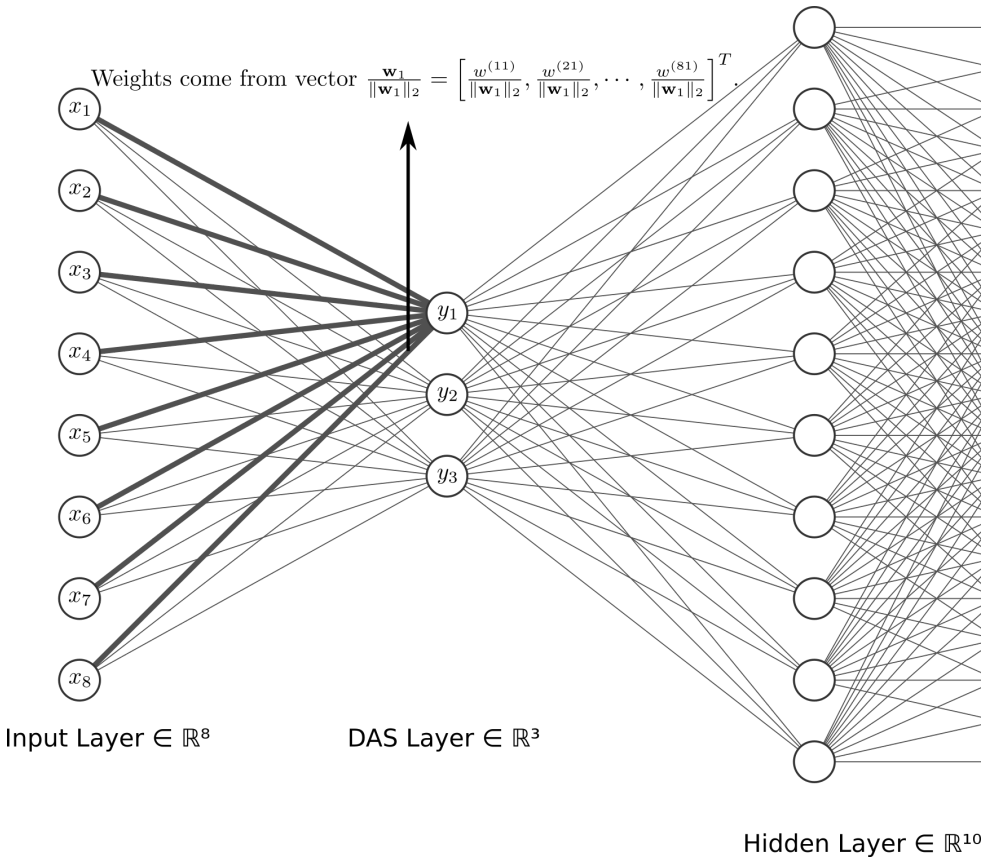

Figure 7: The input layer (with inputs denoted as x_i here), and the Deep-Active Subspace layer with orthonormal weight matrix W_1. This weight matrix is parameterized by Gram-Schmidt orthogonalization.

2.3 Advanced Uncertainty Quantification Patterns

The UQ analysis of a multiscale simulation is usually implemented using a non-intrusive method such as a Monte Carlo method, polynomial chaos expansion (PCE), or with surrogate modeling techniques. By exploiting the computational structure of multiscale simulation, the cost of a multiscale UQ can be reduced by relying on a family of recently proposed semi-intrusive UQ methods [10]. We extend the notion of exploiting generic computational structure of multiscale simulations to further improve the

computational efficiency by identifying generic Uncertainty Quantification Patterns (UQPs), and then implementing the UQ analysis with these UQPs. In fact, we propose a series of UQPs according to the degree of intrusion and architecture of multiscale simulation. These UQPs provide basic building blocks for creating tailored UQ for multiscale models. The UQPs are implemented as generic templates, which can then be customized and aggregated to create a dedicated UQ procedure for multiscale applications.

An overview of the UQPs is shown in Figure 8. We categorize the UQPs according to the degree of intrusion and architecture of multiscale simulation. The first category, UQP1, is a pattern which represents the commonly used non-intrusive methods. UQP2 and UQP3 are based on semi-intrusive methods and apply to acyclic and cyclic multiscale models, respectively. In addition, we consider two ways to optimize computational efficiency: efficient sampling (A) and surrogate modeling (B). Efficient sampling refers to sampling techniques more efficient than basic methods and hence reduces the number of samples required to perform UQ for the given set of input parameters. As explained before, surrogate modeling refers to replacing the computationally expensive model or submodel by a surrogate which approximates the behavior of the original model at lower computational cost.

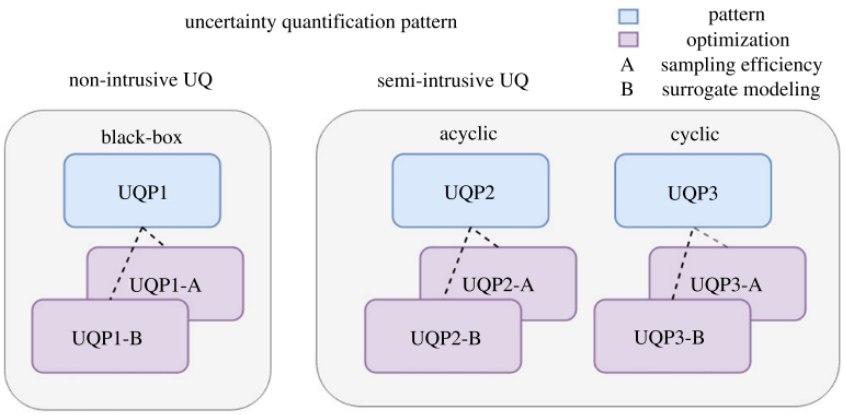

Figure 8. The summary of uncertainty quantification patterns. UQP1 is taken as a black-box pattern where the general non-intrusive UQ methods can be applied. UQP2 and UQP3 are based on the semi-intrusive UQ methods which exploit the coupling in multiscale simulations.

2.3.1 UQP1: Non-intrusive pattern

Consider a prototypical multiscale model consisting of two submodels, F and G, coupled together in the most general sense, either acyclic or cyclic, as shown in Figure 9 by the arrows in between submodels. Both submodels F and G take uncertain inputs, e.g. initial conditions, boundary conditions or model parameters (shown by blue incoming arrows) and both produce QoIs with uncertainties (the red outgoing arrows). The simplest UQP, called UQP1, does not exploit the

multiscale simulation structure, and treats the multiscale model as a black box that has inputs and produces outputs. As shown in Figure 9, the UQ is performed by using non-intrusive methods on the application as a whole, and quantifying the uncertainty relative to a QoI that is part of the final application output.

To optimize the computational efficiency of UQP1, one can apply advanced sampling methods to reduce the total number of runs of the simulation (UQP1-A, e.g. the dimensions-adaptive SC or deep active subspaces described earlier). In addition, a low-cost surrogate model can be built based on the mapping between uncertain inputs and the QoI to replace the model as a whole (UQP1-B). Although this is the most common way to implement the UQ analysis for a computational model, the method can be computationally prohibitive for many multiscale applications that require significant computational resources. Therefore, in the next section, we discuss how the multiscale structure of such applications can be exploited in order to perform uncertainty propagation more efficiently.

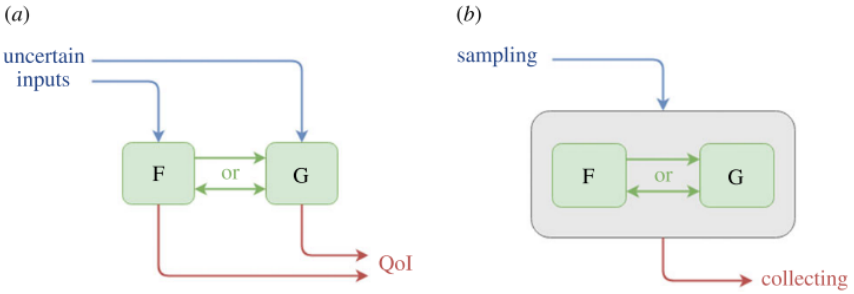

Figure 9. (a) Prototypical multiscale model. F and G are two submodels of a multiscale simulation coupled in a general sense. Both submodels take uncertain inputs and output QoIs with uncertainties (b)UQP1 considers the multiscale model a black-box that has inputs and produces outputs.

2.3.2 UQP2: Semi-intrusive acyclic pattern

According to the coupling topology of the multiscale simulation, a main distinction can be made between acyclic and cyclic multiscale models. In the case of acyclic structure, uncertainty propagates in one direction through the multiscale model. Output uncertainty of one single component creates input uncertainty of another component. UQP2 performs non-intrusive UQ on consecutive single components in an acyclic model (Figure 10).

There are two important advantages of applying UQP2: transparency and efficiency. UQP2 makes it possible to investigate how uncertainty propagates and amplifies within each component of the model. UQP2 can be realized as a sequential application of UQP1 (Figure 10). After applying UQP1 to submodel F, the data to be sent to submodel G has now turned into an uncertain output, which is

then converted into uncertain input for submodel G. Therefore, UQP2 makes uncertainty propagation more transparent and provides additional information on uncertainty as it propagates between submodel levels.

The second important advantage of UQP2 is that it introduces different ways of improving the computational efficiency of corresponding UQ analysis. One way to obtain better efficiency is to apply resampling: samples of the output of one submodel can be used to approximate the probability density function (PDF) of this output. Then, this output can be considered along with other uncertain inputs of the next submodels and, therefore, uncertainty propagation of this submodel can be performed independently of the analysis of the previous one. This allows applying the most efficient uncertainty propagation methods for each of the single-scale models. This also allows for a more flexible implementation of UQP2-A, in which sampling for each submodel can apply a different advanced sampling method. Another way to improve the efficiency for acyclic multiscale models is to build a surrogate model of the most expensive single-scale model.

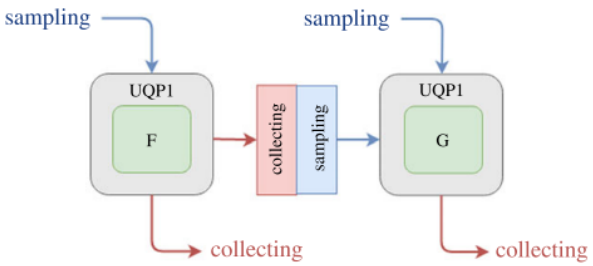

Figure 10. UQP2: Semi-intrusive acyclic pattern. UQP2 performs non-intrusive UQ on submodels. After applying UQP1 to submodel F, the data to be sent to submodel G has now turned into an uncertain output which then is converted into uncertain input for submodel G.

2.3.3 UQP3: Semi-intrusive cyclic pattern

For cyclic multiscale models, we propose UQP3 as shown in Figure 11. UQP3 again performs non-intrusive UQ on individual single components. Compared to the UQP2, in UQP3 we add the Coordinator module between the submodels to orchestrate the data flows. For UQP3, resampling cannot be applied as it can be for acyclic UQP2 workflows, because the models alternate to modify the model's state, which causes their states and parameters to become correlated as we explained. The output of the first model must be sampled conditionally on the value of the shared parameters for the target second model instance. Therefore, instead of resampling, the semi-intrusive Monte Carlo method can be applied to improve the efficiency (UQP3-A). Additionally, it is important to note that the implementation of UQP3-A is not limited to such Monte Carlo methods. Other UQ methods,

such as PCE or stochastic collocation, can be applied to reduce its computational cost and improve the efficiency of the multiscale UQ analysis.

For UQP3-B, again each submodel can be replaced by a surrogate model that produces an approximation significantly faster than the original submodel. In a time-scale separated coupling, the micromodel will run many more time steps than the macro model, so it is usually the target for surrogate modeling. A distinction can be made between stateful and stateless submodels. In a stateful case, the output of the micromodel depends on all previous inputs. Timescale overlapping couplings are similar to timescale separated couplings with a stateful micromodel, but will not be discussed further here.

Besides, the Coordinator in UQP3 allows us to implement UQ with an online surrogate model. Instead of deploying a pre-trained surrogate model, one can dynamically and adaptively train and test the surrogate model during the UQ implementation. Such a process is similar to the concept of active learning [11], but embedded in a UQ procedure. This concept of online surrogate model can be an interesting research topic to further improve the computational efficiency of UQ.

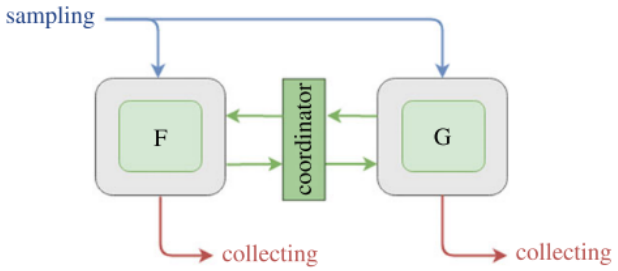

Figure 11. UQP3: Semi-intrusive cyclic pattern. UQP3 again performs non-intrusively on consecutive single components. Compared to UQP2, UQP3 has an additional box (Coordinator) between the submodels, to orchestrate the subsampling, interpolation and statistical testing of the interpolation.

2.4 Multiscale Verification & Validation Patterns

Within VECMA we have established four patterns for the verification and validation of single- and multiscale models. We call these Verification & Validation Patterns, or VVPs. As a reminder to the reader, verification concerns the correctness of a solution relative to a given model (computational or mathematical), whereas validation concerns the correctness of that model relative to the real-world system it is intended to describe [12].

The four patterns we have established do not exhaustively cover the space of application-agnostic verification and validation activities. Instead, they are intended as a starting point for developing

further pattern-based generic V&V (as planned for instance in the recently awarded UKRI-funded SEAVEA project).

These patterns include:

3. Stable Intermediate Form (SIF)
4. Level of Refinement (LoR)
5. Ensemble Output Validation (EOV)
6. Quantities of interest Distribution Comparison (QDC)

2.4.1 Stable Intermediate Form

Stable Intermediate Forms (SIF) is a general, gradual, observable and risk-averse pattern for making progress in the simulation development iteratively and limiting the effect of failure. When undertaking a series of changes to move from one step to another, each intermediate step is evaluated and once it is considered stable it will be used as the basis for further changes. This helps in gaining confidence in the iterative and incremental model development rather than carrying out sudden changes with a single large stride, making it too complex and possibly deviated from the development pipeline.

To apply this VVP we need to perform the SIF code and the incremental code over a representative range of problems. Replicated execution may be required for these executions to reduce the effects of aleatoric uncertainty.

Requirement: the simulation needs to be sufficiently mature that a previous SIF is indeed in existence and accessible.

Typical inputs:

- A representative set of simulation problems.
- The stable (SIF) code.
- The code to be verified.
- A metric denoting the error.

Typical output:

- Boolean: the code is verified successfully if it performs with the same or lower error than the SIF.

2.4.2 Level of Refinement

Level of Refinement (LoR) is a general verification pattern that seeks asymptotic behavior in Quantities of Interest upon increasing the resolution of certain model parameters. Here, the same QoI is computed at every given resolution. One might for instance perform an error convergence study in Computational Fluid Dynamics, where the same output quantity is computed on grids of decreasing spatial cell size and time step size. The point from which the simulation outcomes no longer change significantly marks the largest space (or time) resolution for which the results are independent of the computational grid. This application is commonly performed in the scientific community, and not explicitly presented here.

However, another and less common example is demonstrated below: this concerns the convergence of Sobol sensitivity indices when increasing the polynomial order. Sobol's method is a common technique for performing sensitivity analysis, and we can apply this VVP to investigate whether the error decreases with polynomial order, or whether overfitting is a risk and could lead to higher error for high polynomial orders.

Requirement: the simulation needs to be able to produce results across different levels of refinement, e.g. across different resolutions, time steps, or different numbers of convergence iterations.

Typical inputs:

- Configurations that are identical aside from a different level of refinement.
- The simulation code.
- A metric denoting the (validation) error.

Typical output:

- Numerical: an aggregated metric that represents the validation outcome, which is inspected for convergence with the level of refinement.

Level of Refinement in computing Sobol sensitivity indices

Sobol indices are global, variance-based sensitivity measures, which measure the sensitivity of the simulation output with respect to the input parameters, when independent probability density functions are prescribed for the latter. Computation involves some kind of quadrature method, since

multiple integrals are involved in their formulation. Several methods exist to address this problem. One possibility is to replace the original simulation code with a Stochastic Collocation (SC) surrogate model, in which case the Sobol indices can be approximated in a post-processing step, see [13] for details. For the purpose of the current discussion, it suffices to say that the sensitivity indices are now dependent upon the chosen polynomial order of the SC surrogate. This polynomial order p regulates the resolution of the sampling plan in the input space, i.e. the points in the input space at which the actual (potentially expensive) simulation code must be evaluated in order to build the SC surrogate. This problem is therefore a natural example for a LoR verification. Our goal is to examine the convergence of the Sobol indices with increasing p (increasing number of code evaluations).

2.4.3 Ensemble Output Validation

Validating (multiscale) simulations against observational data is essential to establish their credibility, and more often than not this is done in an *ad hoc* manner against only a small number of experimental/observational results. The Ensemble Output Validation (EOV) pattern is intended to help improve this, by providing a shorthand technique to facilitate rapid validation of simulations across a wide (and potentially diverse) range of different settings.

Given an ensemble of validation/verification output directories, the EOV pattern will operate a sample testing function on each director, print the outputs to screen, and use an aggregation function to combine all outputs into a compound metric.

Requirement: the simulation needs to be able to be validated across a comprehensive set of different problems (or situations).

Typical inputs:

- Configurations that represent a comprehensive set of validatable problems.
- The simulation code.
- A metric denoting the total validation error.

Typical output:

- Numerical: an aggregated metric that represents the validation outcome.

2.4.4 Quantities of interest Distribution Comparison

Often validation does not depend on single QoI values, but on a distribution of QoI values that may be non-trivial to characterize. The Quantities of interest Distribution Comparison (QDC) VVP serves to provide a tool to compare these distributions of QoIs with those obtained from experimental/observational data.

The QDC VVP extracts a distribution of QoIs from the simulation runs (using the decoder + analysis). It applies a similarity measure to quantify the similarity between the QoI distributions from the simulation, and from the validation data.

Requirement: the simulation needs to have uncertainty in its results, that is, it produces a distribution of QoIs rather than singular quantities.

Typical inputs:

- Configurations that represent a comprehensive set of validatable problems.
- The simulation code.
- Means to extract the QoI.
- A similarity measure.

Typical output:

- Numerical: an aggregated metric that represents the validation outcome.

2.5 Implementation of UQPs and VVPs in VECMAtk

The main reason for implementing the UQPs and VVPs in a software toolkit is to showcase the practical benefits, both in terms of reduced development effort and in newly enabled VVUQ procedures, of these patterns. Because the VECMA toolkit (VECMAtk) consists of a diverse set of components, each of which comes with its own development philosophy, we have chosen to facilitate the implementation of the patterns using four different approaches. In this section we will summarize these approaches, and indicate which approach has been used for which type of pattern and tools, and why.

Approach #1: Integration of pattern functionality in application-agnostic tools. A large number of UQPs have commonalities in terms of implementation requirements. Many of them rely on generic,

conceptual building blocks such as sampling, encoding, decoding, and training / applying surrogate models. In particular, within EasyVVUQ, QCG-PilotJob and EasySurrogate we have chosen to offer the pattern functionalities through the implementation of these building blocks in the software. The user is then able to use a range of UQPs by combining these building blocks. How to do this is extensively explained as part of the VECMAtk documentation, for example within the wide range of EasyVVUQ tutorials.

Approach #2: Direct mapping of patterns to implementation. For some software components, in particular MUSCLE3, it has been possible to directly map the patterns from formalism to implementation. MUSCLE3 relies directly on the Multiscale Modeling and Simulation Framework, and as a result this tool directly supports a wide range of UQPs, which are exposed through a range of published examples and tutorials [14].

Approach #3: Encoding of patterns in tool- and application-agnostic code templates. The VVPs introduced in VECMA serve to capture a range of essential V&V activities for a diverse range of applications and use cases. Because many of the VVPs are less reliant on advanced workflow and coupling approaches, it has been possible to capture several VVPs in the form of application-agnostic code templates[1]. These code templates are currently integrated within FabSim3, but have virtually no dependencies on external code and can therefore be easily taken out and reused in other software components. Making these patterns lightweight, generic and easily reusable has been a focus over the course of VECMA, and in the follow-up SEAVEA project (a UKRI-funded initiative that launched in August 2021) we aim to demonstrate the reusability of these patterns by also integrating them into other tools.

Approach #4: Ad-hoc implementations of patterns in specific applications. The VECMA toolkit has seen its main release in M36, and has found uptake across diverse communities. Yet there are a few patterns that have not been implemented completely in an application agnostic manner. This is in particular the case for VVP #1, Stable Intermediate Form, which has been implemented in an ad-hoc manner for the epidemiology application, but still has to be generalized using approach #3. Our ambition is to have all ad-hoc implementations replaced with generic implementations (i.e. that follow the first three approaches) by the end of the project.

To summarize, we provide a listing below as to which tools support which UQPs and VVPs:

[1] see e.g. https://github.com/djgroen/FabSim3/blob/master/fabsim/VVP/vvp.py

UQP	**Tool(s) supporting it**
1	All
2	EasyVVUQ,MUSCLE3
3	EasyVVUQ,MUSCLE3
4	MUSCLE3, FabSim3
A (sampling efficiency)	All (this has grown to be a core requirement in VECMA)
B (surrogate modeling)	EasySurrogate,MUSCLE3
VVP	**Tool(s) supporting it**
Stable Intermediate Form	FabCovid19 (FabSim3 implementation in progress)
Level of Refinement	FabSim3
Ensemble Output Validation	FabSim3
Quantities of interest Distribution Comparison	EasyVVUQ

Table 1: Mapping of patterns to tools supporting their implementation and use.

2.6 Scalable UQPs for current petascale and emerging exascale architectures

The scalability of UQPs, and more generally of VECMAtk, when defining, implementing and deploying multi-scale simulations VVUQ has been addressed at different levels, from theory to practice. In this section, we present the work done toward scalable UQPs. This includes the ability to run large ensembles on HPC resources. We have developed various components in the VECMAtk that can handle large ensemble sizes, e.g. FabSim3 or the QCG tools. That said, many UQ problems are subject to the curse of dimensionality (an exponential increase in computational cost with the number of uncertain inputs), which simply cannot be tackled with brute-force computational power alone, even with exascale resources. One clear example of this is our Molecular Dynamics study [7]. Therefore, the second pillar of scalability we devoted attention to is “methodological” scalability. This involves bringing down the computational cost of the method. We believe that in most (future) case studies, performing UQ on applications that involve expensive computational models with real-world significance will require a combination of HPC and methodological scalability. For a presentation of the efficient use of HPC infrastructure through VECMAtk components, please refer to our deliverable D5.3. We briefly describe methodological scalability below.

2.6.1 Scalability of the simplest UQP: UQP1

Scalability of the UQPs and of the VECMAtk has been considered and addressed at a number of different levels and by most of the Work Packages.

WP2 has devised a number of approaches that reduce the computational requirements from the initial methods. As an example of this, Sobol indices which characterize the source of output variance tracing it back to the inputs, can be calculated with pure Monte Carlo methods. Polynomial Chaos Expansion (PCE) and Stochastic Collocation (SC) both have a better scaling of the accuracy of the Sobol indices with regard to the number of samples (under some often satisfied assumptions).

WP3 then implemented these methods in the EasyVVUQ component of the VECMAtk, and were used by WP4. Then, in a collaborative effort, WP3 and WP4 looked at the scaling of the implementation and identified bottlenecks and unexpected scaling. As an example of this, the initial implementation of the PCE analysis phase took an unexpectedly large amount of time -- which was fixed by changing the method for calculating the Sobol indices. Similarly, the scaling for setting up the cases to be was also identified as a problem, and fixed.

However, even with these improvements, the time needed for high dimensional problems remained a problem, with a scaling of N^2 or higher in the calculation of the Sobol indices. While the time taken could be reduced by parallelization, the problem will remain. WP2 then devised, and WP3 implemented, an extension of the techniques to use dimensional adaptation to reduce the number of samples required, calculating these at positions that are optimized. These changes have allowed WP4 to extend their investigations of uncertainty to many more varying parameters. As examples of this, this allowed for UQ to be performed for cases with 19 varying parameters in the COVID simulations [6], and as shown in D4.4 for the fusion tutorial case, the same accuracy could be achieved in 1/160th of the time when applying the dimension adaptive approach instead of the usual SC case.

WP5 together with WP3 played a key role in much of the above work by optimizing the ability for running a large number of cases, details of which are presented in D5.3.

2.6.2 Scalability beyond UQP1

The various surrogate modeling strategies aim to bring down the computational cost of multiscale simulations (UQP3-B). We refer to the stochastic surrogates, reduced surrogates and surrogates based

on convolution neural nets of D2.2. The reduced surrogates also compress the size of the training data by several-order of magnitude, thereby circumventing potential memory constraints.

Further, the various surrogate modeling strategies aim to bring down the computational cost of multiscale simulations (UQP3-B). We refer to the stochastic surrogates, reduced surrogates and surrogates based on convolution neural nets of D2.2. A separate journal article was written that examines the scalability of all UQPs, see [14]. The stability of coupled physical - surrogate systems is currently investigated, e.g. via online learning as discussed above. A detailed discussion about the coupling issue of online learning can be found in Appendix 4.2. Besides, the modularization based on muscle3 (Appendix 4.1) is the key to the implementation of online learning. It allows light assembling of surrogates to any submodels.

The reduced surrogates also compress the size of the training data by several orders of magnitude, thereby reducing the memory requirements. For instance, for the 2D ocean model considered in [3], the size of the data (for each snapshot in time), was reduced from 4096 points (64 x 64), down to just 2 points. As we may need a large number of snapshots to train a surrogate, the memory savings are significant. Moreover, the savings are even more pronounced in the case of 3D models, where each snapshot may for instance consist of 64 x 64 x 64 points. That said, the main point of these reduced surrogates is that the size of the surrogate itself is smaller, as we only need to learn a surrogate in the reduced dimension (e.g. 2), instead of the full physical (spatial) dimension (e.g. 64 x 64 or higher).

3. Conclusions

In the deliverable, we have presented advanced algorithms for multiscale VVUQ (Verification, Validation and Uncertainty Quantification), Uncertainty Quantification Patterns (UQPs) and Verification & Validation Patterns (VVP) for multiscale simulations. We have completed the corresponding planned implementations of the UQPs and VVPs in the VECMAtk.

4. Appendix

4.1 Implementation of UQPs using MUSCLE3

We describe how the UQPs can be implemented using the Multiscale Coupling Library and Environment (MUSCLE 3) [15] in this section. MUSCLE 3 is a coupling framework that is designed for coupling temporally and/or spatially scale-separated multiscale models. In a MUSCLE 3 simulation,

submodels run simultaneously as separate programs and exchange information via the network, through the libmuscle library. Each submodel has one or more ports, which are named connectors through which it sends and receives messages. The ports of the submodels are connected by MUSCLE according to a description in a configuration file based on an extended version of the Multiscale Modeling Language [16]. The configuration file also contains model settings (parameters and other configuration), and MUSCLE 3 has a mechanism through which one model component can send new settings to another component. This settings overlay overrides global settings, and is automatically propagated to subsequent connected components. Many copies of a (sub)model can be instantiated, which combined with the settings overlay allows for implementing UQ, as described below.

UQP1: The UQPs are flexible with respect to which specific UQ methods are used. In this section, we use plain Monte Carlo as an example, as it is simple and well known. The Monte Carlo method for performing a forward UQ comprises three steps: (i) sampling the uncertain parameters n times, (ii) running the model once for each sampled value and (iii) calculating statistics of the resulting set of the outputs. For a multiscale model coupled using MUSCLE 3, this can be implemented by adding two additional components, a Sampler which samples the uncertain parameters, and an Analysis component, which calculates statistics, and connecting them to n instances of the original model (Figure 12a). The Sampler produces n samples and sends them to its samples_out port. This is a vector port (shown by the square brackets), which is used to connect to a set of submodel instances, rather than to a single instance. It is connected to the muscle_settings_in scalar port of the existing model. This port implements the settings overlay mechanism described above, so that each instance of the original model will run with the corresponding parameters. The output of the model runs is then sent to the Analysis component, which receives both the results and the parameter overlay for each instance, thereby furnishing all the information it needs to calculate the required statistics.

In order to implement UQP1-B, a surrogate model needs to be constructed, and substituted for the original model. This can be done by changing the configuration file to wire in the new component instead of the original model.

UQP2: In UQP2, we apply UQP1 to each of two models coupled sequentially (Figure 12b). The Sampler and Analysis components of UQP1 are reused here. Alternatively, two separate Analysis modules can be used, one for each submodel. In this case, only marginal distributions and correlations among each submodel's QoIs can be estimated, but not correlations between the different submodels' QoIs. The submodels remain unchanged, as in UQP1. To allow the use of different ensemble sizes or UQ methods for the two submodels (UQP2-A), a Resampler component is

added in between. It receives n outputs of the first submodel on its vector port in, and converts those into m inputs for the second submodel, which it sends on its vector port out. Like the Analysis component, the Resampler receives both the first submodel's output and its settings overlay. It is used to produce a new set of settings and corresponding inputs, and sends those on to the second submodel. Using surrogate models for individual submodels (UQP2-B) can be done just as for UQP1 by changing the configuration to substitute a surrogate for the component.

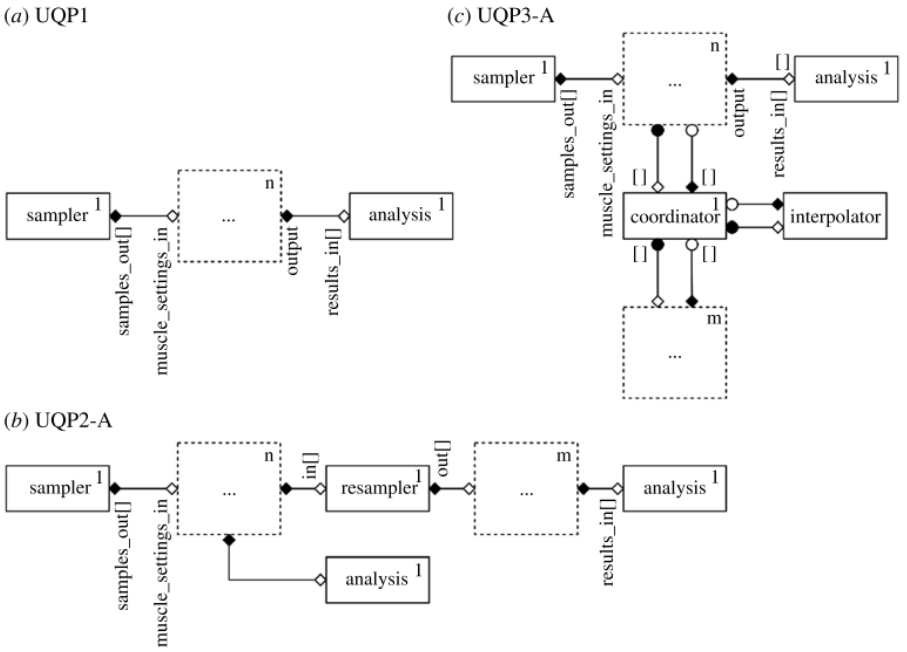

Figure 12. Implementing UQPs using MML and MUSCLE 3: (a) UQP1, (b) UQP2-A and (c) UQP3-A. Boxes represent components with their number of instances in the top-right corner, dashed boxes are placeholders for submodels, which are substituted into the pattern. Lines denote conduits through which data may be sent. An open diamond receives initialization data, a closed diamond sends the final result, a closed circle sends an intermediate output at each time step, and an open circle receives state or boundary conditions at each time step.

UQP3: Cyclic coupled models of time-scale separated processes consist of a macro model that at each time step calls a micromodel, which runs to convergence and sends a result back to the macro model. To implement UQP3, we can first apply UQP1 by connecting the Sampler and Analysis components to the macro model, and instantiating n copies of both submodels. Here too parameters will be automatically propagated to the micromodel, and both submodels remain unchanged.

For UQP3-A, the algorithm is generic, but the specific interpolation function used is model-specific. We add the generic Coordinator component, which implements the generic algorithm, but attach to it a model-specific Interpolator, which interpolates the micromodel output in a model- specific manner (Figure 12c).

The Coordinator component, being wired in between the two sets of submodel instances, intercepts messages from each ensemble member, including the parameters for that ensemble member. Messages from the initial m macro model instances are sent on to their corresponding micromodel instances, and the replies are then passed back. Thus, these ensemble members run normally. However, the micromodel outputs and their corresponding parameter settings are also sent to the Interpolator, which stores them. The remaining n − m messages and parameters are not forwarded to the micromodel, but instead the parameters are sent by the Coordinator to the Interpolator, which interpolates the previously stored messages to produce estimates of the micromodel output for the remaining n − m parameters. It sends these back to the Coordinator, which forwards them on to the corresponding macro model instances. Thus, the remaining n − m ensemble members use interpolated values, saving this many runs of the micromodel. Once micromodel results have been sent for all ensemble members, the macro model proceeds to its next time step. A more advanced version of the algorithm adds a validation step and dynamically adds more actual micromodel runs until the Interpolator returns results of sufficient quality.

For UQP3-B, a (pre-trained) surrogate can be substituted for the micromodel by changing the configuration. Training a surrogate while running can be done in a similar manner, substituting a surrogate model for the Interpolator. This requires the Coordinator to work slightly differently, sending micromodel input/output pairs, rather than parameters/output pairs. If the micromodel is stateful, then its output at time t depends on all messages sent to it at previous simulation time steps. In this case, the messages must be accumulated either by the Coordinator or by the surrogate model to produce a good prediction.

4.2 Study on model and surrogate coupling

Two major variations of coupling have been investigated by Atos to identify the performance bottlenecks that may appear.

4.2.1 Strong coupling

Here, the coupling is performed primarily via API calls. This requires a well-defined interface between the surrogate and encapsulating code/model. Thus, modifications in the sources of the original code are necessary, making this method intrusive.

During testing of this coupling method, it became apparent that the best results are obtained by low-level languages like C or Fortran for the surrogate model (assuming that the physical model is also written in C/Fortran). Incorporating Python in the loop, for example to interface to TensorFlow, significantly decreased the performance by a factor of up to four. This is due to necessary copy operations between Fortran and Python to ensure that Python arrays are Fortran-friendly. Also, there is no immediate benefit in this simple case study for using a GPU, as the model has to be reloaded. However, this is done only once at the initialization step, as we were able to demonstrate.

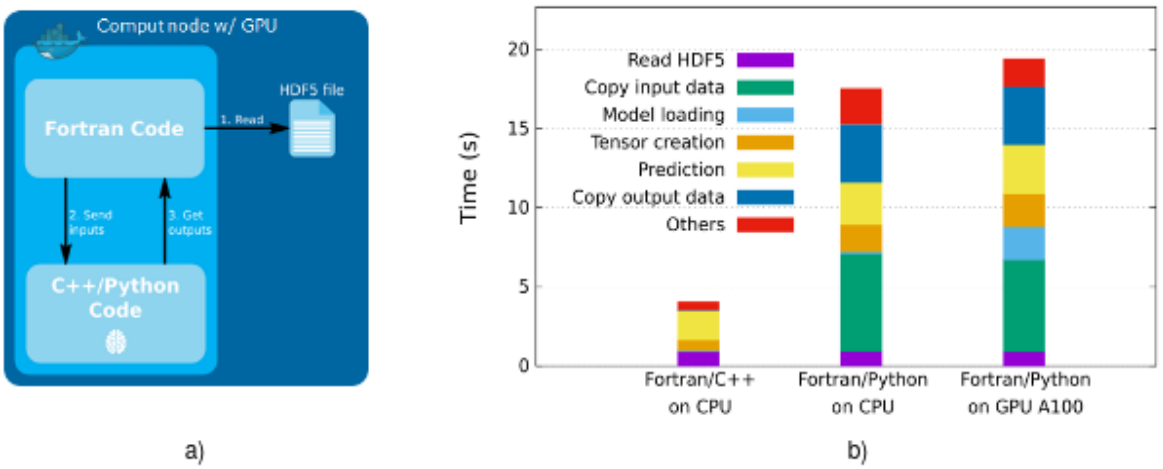

Figure 13. a) Schematic diagram for strong coupling between surrogate model and original model. b) The corresponding computational cost (time) of each component.

4.2.2 Weak coupling

On the other hand, the non-intrusive method of coupling is defined as weak coupling. Here the lifecycle of the original model is kept unchanged. Intermediate data exchange mechanisms between the codes (MPI, shared memory, or over TCP) are used to “transport” the data.

As a prototype, three docker images have been created to two separate nodes. On node 1, the master image running mpirun and a slave image which executes the physical model are set up. On the other node, a slave image containing the Python AI model is spawned. This physical separation of nodes allows for the possibility to choose a GPU accelerated node for the AI model.

Having all three images running, communication between the Fortran code (which runs the physical model) and the Python code via MPI is possible.

On a supercomputer, arguably the most efficient and flexible way to communicate goes via message passing which properly exploits the existing hardware (e.g. InfiniBand). As shown above

demonstrated a coupling method across multiple docker images where communication is handled by OpenMPI. Alternatively, Muscle3 is a candidate (probably best with the MPI backend, to ensure that it scales to a larger number of nodes).

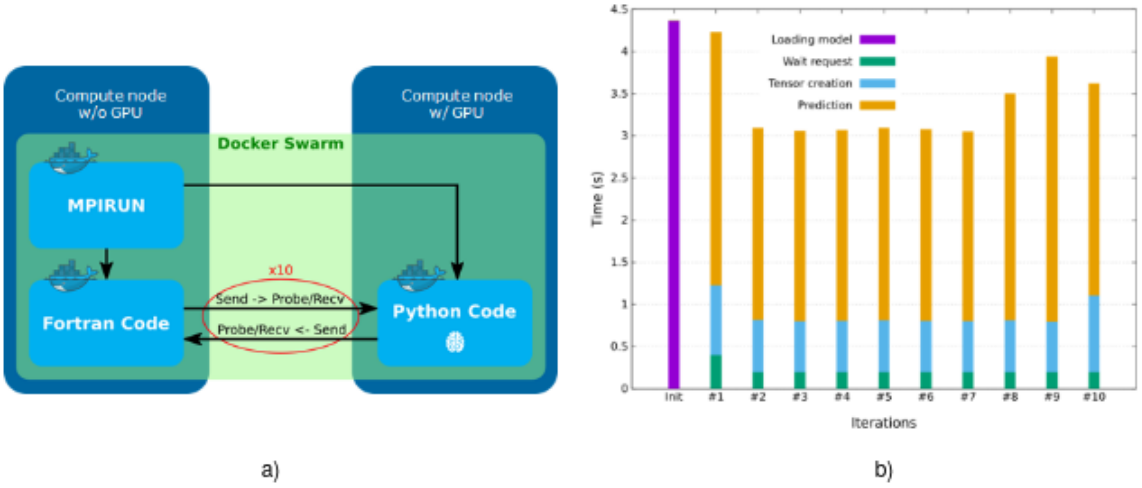

Figure 14. a) Schematic diagram for weak coupling between surrogate model and original model. b) The corresponding computational cost (time) of each component.

4.3 Generic scalable workflow including online learning

The process of online learning has been independently looked at by Atos's developers. Figure 15 presents a generic workflow.

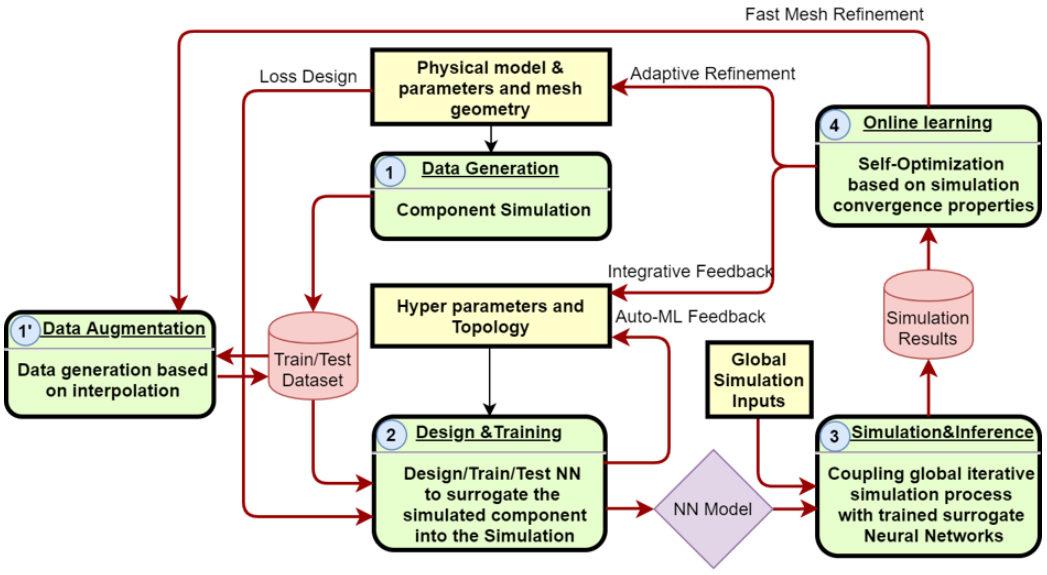

Figure 15. The generic workflow of online learning

To ensure that all semantics used in this figure are well-defined to following list gives a brief overview:

- Agent: a neural network acting as an inference engine, surrogating a component in the global simulation (the associated environment).
- Topology: the structure of the agent, e.g. depth, width, activation functions, dense or convolutional layers...
- Weight parameters: the weights and biases, tuned for a fixed topology by gradient-based methods.
- Hyper-parameters: training session parameters, e.g. learning rate, number of epochs, optimizer...
- Operator: operation on an agent's topology and parameters used to navigate the search space, e.g. backpropagation learning, crossover of two networks...
- Online learning: iteratively improve a model presented with new data of the same task.
- Meta-learning: learn an optimal structure (topology, weights...) of a meta-learner to perform better during training on new tasks, learning to learn paradigm.
- Auto-ML: any method resulting in the automatic tuning of a model's parameters, often used as a short-loop improvement step.

Similar to the workflow proposed by VECMA, data is generated from initial runs of the physical model (step 1). In step 1', we merely propose to enrich the initial data set with standard ML techniques like data augmentation or the generation of synthetic data (by interpolation, for example) as generating the data can be very costly. Combined, they make up the initial training set. While training the surrogate model (steps 2 and 3) and comparing its performance to the physical model, we can allow for hyperparameters (e.g. learning rate) or topology tuning automatically via Auto-ML feedback. Finally, results from the online training step are fed into the model training but we propose that it can also be used to adaptively refine the computational domain of the physical model. This helps to generate data at a higher resolution in critical areas of the simulation for the next loop iteration.

There are multiple options to implement the online learner. For comparison, they are weighted with these metrics in mind:

- Model agnosticity: the learner should consider the models it optimizes as black boxes. We can decently accept a weaker version of this constraint by providing gradients of the surrogate model (grey boxes).
- Robustness: the learning should be easy to tune and converge towards a local optimum.
- Reflectivity: the learner shouldn't add more parameters to tune than it solves for.
- Exploration/Exploitation trade-off: the learner should balance exploitation of acquired information (interesting regions to refine) and exploration of new regions.

- Online training: the learner should be able to learn from new information while retaining the one from previous examples.
- Complexity: the learner's class of complexity should remain tractable.

Gradient-based optimization

This method is the basis of most prominent algorithms implemented in scipy. However, it can be difficult to use, if the target function is noisy or non-smooth, which makes evaluating the derivatives difficult.

Bayesian optimization with Gaussian process regression (Kriging)

Here, interpolation between the evaluation points is done by a gaussian process [17]. Its mean function corresponds to the prediction made. This method has a tolerance to noisy evaluations, which do not exactly represent the original model, like synthetically generated training data would do. This process is also referred to as “kriging”.

NeuroEvolution of Augmenting Topologies (NEAT)

The NEAT method [18] would fall under the methods of reinforcement learning, which do not rely on gradient-based descent. The model evolves “genetically” where the agents mutate their own ANN (artificial neural networks). Training these in parallel requires vast amounts of computational power, which is only feasible if the training sets are reasonably small.

However, instead of converging to a more optimal distribution of weights in the neuronal network, NEAT works on the topology of the network instead. This gradually explores the high-dimensional search space instead of tackling it with a full, dense neural network. Innovations in the population are protected by speciation.

	Gradient-based	Kriging	NEAT
Model agnosticism	required derivatives	derivative-free, no information required	requires access to weights and topology of model
Robustness	high	high	high
Reflectivity		Few tunable parameters	self-reflective
Exploration/Exploitation			speciation
Online training		possible	possible
Complexity		cost scales with power of 3 to the number of evaluations	costly

References

[1] Crommelin, D., & Edeling, W. (2021). Resampling with neural networks for stochastic parameterization in multiscale systems. *Physica D: Nonlinear Phenomena*, *422*, 132894.

[2] Rasp, S. (2020). Coupled online learning as a way to tackle instabilities and biases in neural network parameterizations: general algorithms and Lorenz 96 case study (v1. 0). *Geoscientific Model Development*, *13*(5), 2185-2196.

[3] Edeling, W., & Crommelin, D. (2020). Reducing data-driven dynamical subgrid scale models by physical constraints. *Computers & Fluids*, *201*, 104470.

[4] Gerstner, T., & Griebel, M. (2003). Dimension–adaptive tensor–product quadrature. Computing, 71(1), 65-87.

[5] Edeling, W., Arabnejad, H., Sinclair, R., Suleimenova, D., Gopalakrishnan, K., Bosak, B., ... & Coveney, P. V. (2021). The impact of uncertainty on predictions of the CovidSim epidemiological code. Nature Computational Science, 1(2), 128-135.

[6] Gugole F, Coffeng LE, Edeling W, Sanderse B, de Vlas SJ, Crommelin D (2021) Uncertainty quantification and sensitivity analysis of COVID-19 exit strategies in an individual-based transmission model. PLoS Comput Biol 17(9): e1009355. https://doi.org/10.1371/journal.pcbi.1009355

[7] Vassaux, M., Wan, S., Edeling, W., & Coveney, P. V. (2021). Ensembles Are Required to Handle Aleatoric and Parametric Uncertainty in Molecular Dynamics Simulation. Journal of Chemical Theory and Computation, 17(8), 5187-5197.

[8] Constantine, P. G., Dow, E., & Wang, Q. (2014). Active subspace methods in theory and practice: applications to kriging surfaces. SIAM Journal on Scientific Computing, 36(4), A1500-A1524.

[9] Tripathy, R., & Bilionis, I. (2019, August). Deep active subspaces: A scalable method for high-dimensional uncertainty propagation. In International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (Vol. 59179, p. V001T02A074). American Society of Mechanical Engineers.

[10] Nikishova, A., & Hoekstra, A. G. (2019). Semi-intrusive uncertainty propagation for multiscale models. *Journal of Computational Science*, *35*, 80-90.

[11] Liu, H., Ong, Y. S., & Cai, J. (2018). A survey of adaptive sampling for global metamodeling in support of simulation-based complex engineering design. *Structural and Multidisciplinary Optimization*, *57*(1), 393-416.

[12] Oberkampf, W. L., & Roy, C. J. (2010). *Verification and validation in scientific computing*. Cambridge University Press.

[13] Tang, G., Iaccarino, G., and Eldred, M., (2010) Global sensitivity analysis for stochastic collocation expansion. In the 51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference 18th AIAA/ASME/AHS Adaptive Structures Conference 12th, 2922.

[14] Ye, D., L. Veen, A. Nikishova, J. Lakhlili, W. Edeling, O. O. Luk, V. V. Krzhizhanovskaya, and A. G. Hoekstra. (2021). Uncertainty quantification patterns for multiscale models. Philosophical Transactions of the Royal Society A, 379(2197), 20200072.

[15] Veen, L. E., & Hoekstra, A. G. (2020, June). Easing multiscale model design and coupling with MUSCLE 3. In *International Conference on Computational Science* (pp. 425-438). Springer, Cham.

[16] Falcone, J. L., Chopard, B., & Hoekstra, A. (2010). MML: towards a multiscale modeling language. *Procedia Computer Science*, *1*(1), 819-826.

[17] Agnihotri, A., & Batra, N. (2020). Exploring Bayesian optimization. *Distill*, *5*(5), e26.

[18] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99-127.