LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Spatio-Temporal Surrogates for Interaction of a Jet with High Explosives: Part I - Analysis with a Small Sample Size

C. Kamath, J. S. Franzman, B. H. Daub

June 10, 2023

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Spatio-Temporal Surrogates for Interaction of a Jet with High Explosives: Part I - Analysis with a Small Sample Size

Chandrika Kamath, Juliette S. Franzman, and Brian H. Daub

Lawrence Livermore National Laboratory

7000 East Avenue, Livermore, CA 94551, USA

`kamath2,franzman1,daub1@llnl.gov`

9 June 2023

Abstract

Computer simulations, especially of complex phenomena, can be expensive, requiring high-performance computing resources. Often, to understand a phenomenon, multiple simulations are run, each with a different set of simulation input parameters. These data are then used to create an interpolant, or surrogate, relating the simulation outputs to the corresponding inputs. When the inputs and outputs are scalars, a simple machine learning model can suffice. However, when the simulation outputs are vector valued, available at locations in two or three spatial dimensions, often with a temporal component, creating a surrogate is more challenging. In this report, we use a two-dimensional problem of a jet interacting with high explosives to understand how we can build high-quality surrogates. The characteristics of our data set are unique - the vector-valued outputs from each simulation are available at over two million spatial locations; each simulation is run for a relatively small number of time steps; the size of the computational domain varies with each simulation; and resource constraints limit the number of simulations we can run. We show how we analyze these extremely large data-sets, set the parameters for the algorithms used in the analysis, and use simple ways to improve the accuracy of the spatio-temporal surrogates without substantially increasing the number of simulations required.

Contents

1 Introduction

Computer simulations are increasingly being used in science and engineering applications. However, it can be time consuming to run these simulations for a given set of input parameters, especially when the problem being modeled is complex and requires high-performance computing resources. Surrogates, often based on a machine learning model [16], are used to provide a fast, but approximate, alternative that relates the simulation outputs to the corresponding inputs. Such surrogates are relatively easy to create when the simulation inputs and outputs are both scalars. However, when the output is in two or three spatial dimensions and varies with time, relating these spatio-temporal outputs to the scalar inputs becomes more challenging. If, in addition, we are constrained by time or computer resources to generate data for only a small number of simulations, building a surrogate that is accurate, becomes non-trivial.

We describe our work in creating surrogates for a problem in two spatial dimensions, plus time, in two reports. In this first report, we discuss the applications aspect of our work, focusing on issues related to the small number of simulations in our data set. We want to understand whether it is possible to build an accurate, predictive surrogate model under these conditions and to identify simple ways to improve the accuracy of these models. In a companion report [17], we discuss in detail how we made our solution approach tractable, despite the large size of the spatial data generated at each time step of the simulation.

We start this report by describing the problem considered, namely, the interaction of a jet with high explosives, and the two-dimensional outputs generated by the simulations (Section 2). We then discuss the unique aspects of our problem and place it in the context of related work (Section 3). We describe how we address these unique challenges, focusing on ways in which we can improve the accuracy of the surrogate models (Section 4). Using several test cases, we explore how to set various parameters in the algorithms used in our solution approach and the metrics to evaluate the predictions of the surrogate models (Section 5). Finally, we conclude this report with the lessons learned in building accurate spatio-temporal surrogates for a limited number of simulations, each of which generates a large amount of data (Section 6).

2 Description of the problem and data

We illustrate our ideas on building accurate spatio-temporal surrogates using simulation output from a problem describing the interaction of a jet with high explosives (HE). The domain of the problem is a right cylinder with its axis oriented horizontally as shown on the left in Figure 1. There is a steel plate, 1cm thick, near the right end of the cylinder, with the LX14 high explosive to the left of the plate. Both the plate and the HE have a fixed radius of 10cm. A copper jet aligned along the center line of the cylinder, enters the HE from the left. The simulation models what happens as the jet moves through the HE and the plate. The jet is modeled initially as uniform cylinder. It is 10 cm in length with a varying radius. The jet tip velocity is specified as an input parameter; a linearly-varying velocity profile is applied to the remainder of the cylinder
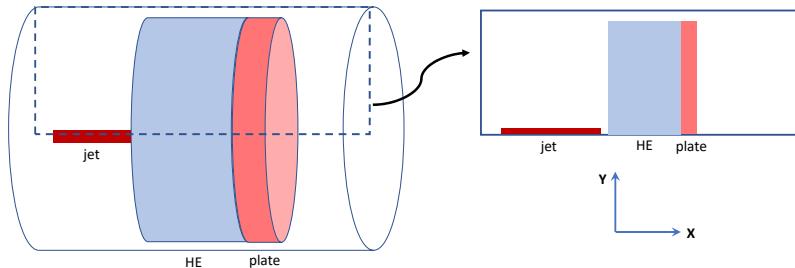

Figure 1: A schematic of the problem being simulated. On the left is the horizontal cylinder in three-dimensions, showing the plate in pink, to the left of which is the HE in blue. The jet, in red, enters the HE from the left. As the problem is radially symmetric about the axis of the cylinder, we need to simulate only the two-dimensional region shown by a dotted rectangle on the left and schematically on the right.

that represents the jet to approximate a stretching metal jet. As the problem is radially symmetric about the axis of the cylinder, only the two-dimensional region shown by a dotted rectangle on the left in Figure 1, and schematically on the right, is simulated.

There are three input parameters for the simulation: the *radius* of the jet, the *length* of the HE to be traversed by the jet, and the *tip velocity* of the jet in the positive x direction. At each time step, the simulation outputs variables of interest, such as mass and momentum, at different points on a grid in the two dimensional region. By running the simulations at select values of these input parameters, and collecting the output at different time steps for each simulation, we can create a data set that could be used to build a surrogate model to predict the output at a new set of input parameters and a given time step. We are interested in determining for example, whether the plate breaks; what is the final position of the plate; and, if the plate breaks, what is the velocity of the jet tip as it comes out on the other side of plate.

To illustrate the instances in our data set, we use four simulations whose parameters are listed in Table 1. Figure 2 shows the output variable, *mass*, at the first and last time steps for these four example simulations. As explained earlier, we have simplified the three-dimensional problem by assuming radial symmetry around the axis of the cylinder, so the output from the simulation is shown as two dimensional images, with the axis of the cylinder shown at the bottom, that is, at $y = 0$. The domain extent in x (along the length of the cylinder) varies as the length of the HE varies across simulations; however, the domain in y ranges from 0 to 11cm for all simulations.

Simulation key	jet radius (cm)	HE-length (cm)	jet tip velocity (cm/μsec)	# time steps	#grid points	outcome
r01_i004	0.15	13.67	0.894	38	2,859,387	almost break
r01_i017	0.17	12.24	0.648	41	2,759,181	no break
r02_i021	0.14	6.77	0.914	30	2,374,179	break
r02_i028	0.23	10.54	0.843	35	2,639,637	break

Table 1: Input parameters for the four example simulations shown in Figure 2. Note the very large number of grid points (over two million) at which variables of interest are output at each time step in a simulation.

In Figure 2, the vertical plate, shown in red, is stationary at time $t = 0$. To the left of the plate is the HE shown in light blue. The jet in shown in red at the bottom of the domain to the left of the HE; it is quite thin relative to the radius of the cylinder, and is barely visible in the images. As the simulation evolves, the jet moves to the right, through the HE, which expands, pushing the plate to the right. At late time, depending on the simulation input parameters, the plate could:

- **break**, with the jet going through the plate and coming out clearly on the other side;

- **almost break**, with the jet either going completely through the plate but barely coming out the other side or the jet going almost all the way through the plate, leaving it barely connected at the bottom;

- **not break**, with the plate remaining attached, either partially or completely, at the bottom. The plate could have moved from its original position at time $t = 0$.

We used the last two time steps in each simulation to assign one of these three class labels to the simulation. This label was not used in building the surrogate; it was used only to ensure we had a good coverage of the design space. We selected the four example simulations in Figure 2 to illustrate these three cases.

In our simulations, the output at each time step consists of the values of variables of interest that are generated at grid points in the two-dimensional rectangular domain. These grid points are on a regular grid, with $\Delta x = \Delta y = 0.0125$cm. There are three variables output: *mass, x-momentum, and y-momentum*; the latter two are shown later in Appendices A and B, respectively. The values of these variables are defined at the center of the cell formed by four nearby grid points. Thus the data appear as an image, with regularly spaced pixels. However, in general, the grid points in a simulation need not be on a regular grid; they could
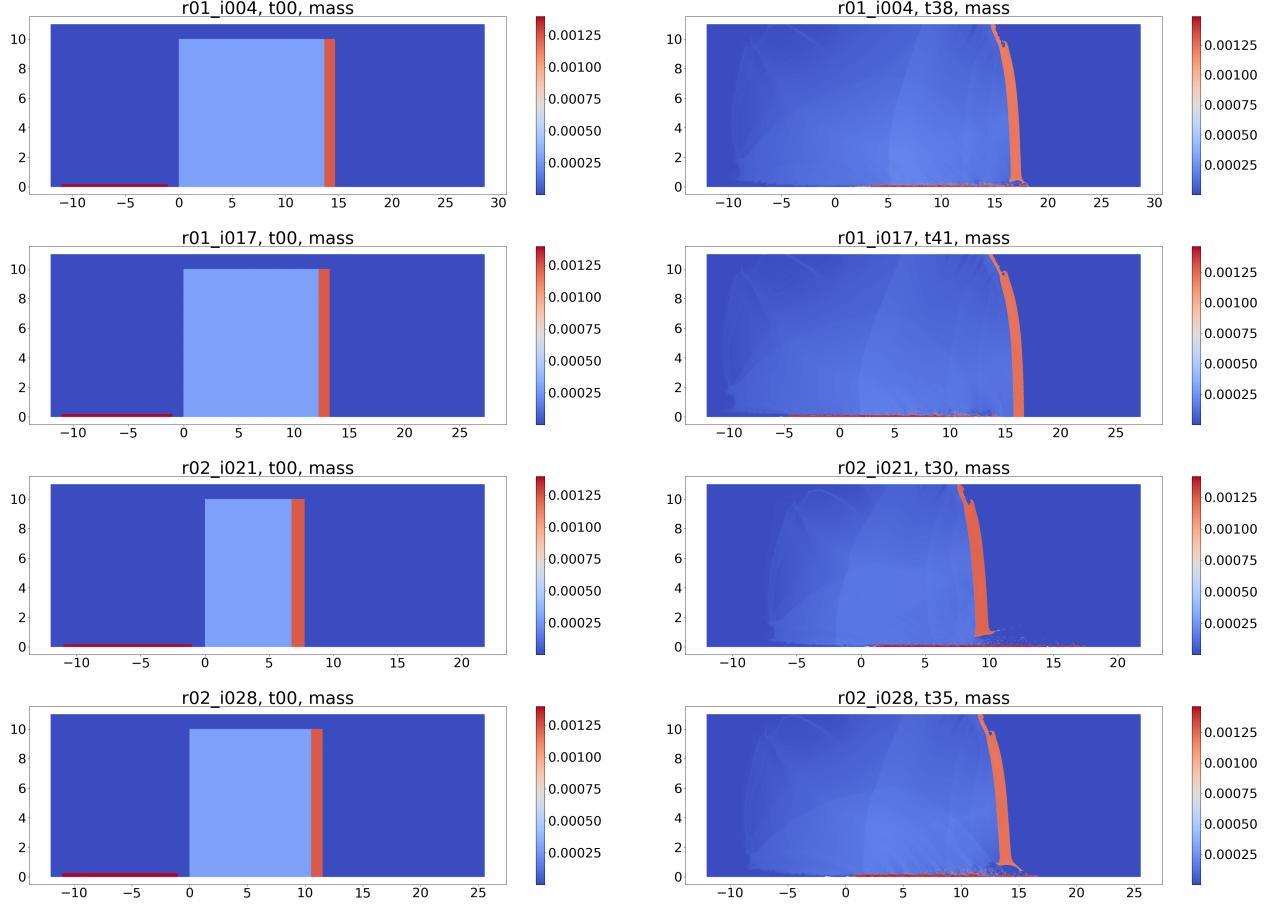
Figure 2: The variable *mass* at the first time step (left column) and last time step (right column) for the four example simulations in Table 1. From top to bottom, simulations with keys r01_i004, r01_i017, r02_i021, r02_i028, illustrate an *almost break*, *no break*, *break*, and *break* case, respectively. The data shown are before the pre-processing steps described in Section 4.2. The vertical red region represents the plate, the light blue represents the high explosive (HE), the horizontal red region at the bottom to the left of the HE at the first time step is the jet, and the dark blue region is air. The right column shows the last time step of each simulation and the effect on the plate as the jet moves to the right, through the HE, and potentially through the plate. The range of x values is different for different simulations, while the range of y values is the same. The HE starts at $x = 0$ in each plot. Note that even when the plate does not break (second row), the plate has moved to the right from its original position.

form an unstructured grid, as in a finite element mesh, or a locally structured grid, as in an Adaptive Mesh refinement (AMR) mesh. As a result, unlike an image, most output from simulations also include the (x, y) coordinates of the grid points. In our work, we retain this association of the coordinates with the grid points as it enables us to extract sub-domains from the larger domain for processing.

Each simulation is run for a fixed number of time steps which is determined as ($\lfloor$(HE-length/jet-tip-velocity)$\rfloor$ + 23), with the output generated at each time step. As both HE-length and jet tip velocity vary with the simulation, the number of time steps also varies across simulations. At early time, as the jet starts to move through the HE, there is little of interest in the simulation output. Once the jet is partway through the HE, as indicated by the first term in the equation above, it starts to influence the location of the plate, until 23 μsec later, it is expected that we should know the final status of the plate. In our work, we consider all the time steps in the analysis; an alternative would be to consider only the later 23 time steps.

The output data for a variable at a time step in a simulation is referred to as a *snapshot*, so named as it is a snapshot of the evolution of the simulation at a particular point in time. For the problem considered

in this report, we generated the data set by identifying sample parameter values in the three-dimensional input space and running the corresponding simulations for the specified number of time steps. For each of the three output variables, the data set consists of the snapshots across all time steps of all simulations.

Our eventual goal in building the spatio-temporal surrogates for the jet-HE interaction problem is to predict what happens to the jet and the plate at late time, specifically, does the jet go through the plate, what is the velocity of the jet tip when it goes through the plate, and what is the location of the plate at late time. However, in this initial study, we limit the scope of the work and explore options for creating an accurate surrogate when we have a small number of simulations. Specifically, we consider how to process large data sets, how to set parameters for the algorithms used in our solution approach, and how to improve the accuracy of the surrogates without substantially increasing the number of simulations required. We evaluate our ideas using a qualitative comparison of predicted outputs for seven test snapshots.

2.1 Challenges to the analysis

There are two main challenges to building spatio-temporal surrogates for our problem:

- The first is how do we build a surrogate that is *accurate*? The predictive accuracy depends on two factors. The first is the quality of the training data. Simulating the jet-HE interaction is resource-intensive, requiring multiple processors of a high-performance computing system. This limits the number of sample points we can run in the three-dimensional input parameter space. These points have to be selected carefully; this is difficult as we do not know *a priori* the outcome of running the simulation at a specific sample point, which in turn implies that we do not know the range of parameter values to use. The second factor influencing the accuracy of the surrogate is the choice of the model used. We want to predict a two-dimensional output, given only four scalar inputs - the three simulation input parameters and a time step. A traditional machine learning model, where we predict scalar outputs for a set of scalar inputs, cannot be used in this case.

- The second, and related, challenge is the very large size of our data set. Though the number of simulations is small, each snapshot has over two million grid points as shown in Table 1. In addition, as the simulations are run on multiple processors, each snapshot is split across multiple files. Any algorithms used to build a high-quality surrogate must be modified to account for both these factors.

This report focuses on the first challenge of building accurate spatio-temporal surrogate models given a limited number of simulations. The second challenge of processing the extremely-high dimensional snapshots generated in a distributed manner is discussed in the companion report [17].

3 Related work

Our approach to creating accurate, two-dimensional, spatio-temporal surrogates builds on some early work in turbulence and pattern recognition, specifically the characterization and recognition of human face images. The early work of Sirovich and Kirby [32, 21] showed that a data matrix, formed by an ensemble of face images, similar to our snapshots, can be transformed using the Karhunen-Loève expansion (similar to the principal component analysis (PCA) [14]) such that each face is written as a linear combination of two-dimensional basis functions, they called "eigenpictures". A close approximation to a face is then obtained by truncating the linear combination to use only a small number of the initial, more important, basis functions and the corresponding weights, thus creating a lower-dimensional representation. Kirby and Sirovich also applied their ideas to problems in fluid flow, including data from simulations, and introduced the snapshot method and the concept of eigenflows [31, 22].

Following this early work, Turk and Pentland [36, 35] showed that these ideas enabled face recognition as we could recognize a new image as a specific face if its weights matched those of the specific face. They referred to the basis functions as "eigenfaces" as they were obtained using eigenanalysis of the data matrix. In later

work, non-linear alternatives were explored to obtain a better representation for data that did not necessarily lie on a linear manifold. The techniques included locally-linear decompositions and neural-network-based auto-encoders, not only for face images, but also speech data in the form of time series and images of handwritten digits [12, 20, 18, 19].

In problems where the data represents output snapshots from simulations run with different input parameters, at possibly different time steps, an obvious next step was to build a predictive model relating the simulation input parameters and time step to the weights characterizing a snapshot. This would enable the prediction of results at parameters not included in the original set of simulations. Such an approach was taken in the early work of Ly and Tran [25], who used proper orthogonal decomposition (similar to PCA) for the decomposition, and spline interpolation for the predictive model. A similar idea was explored by Higdon et al. [11], who also used PCA, but predicted the weights using Gaussian process models, and by Swischuk et al. [33], who compared different machine learning models for predicting the weights.

These ideas have become the subject of much recent research, especially as compute-intensive simulations have become an increasingly important part of design and engineering, requiring rapid generation of results. In particular, in the field on non-intrusive reduced-order modeling (ROM) [28, 29], many options have been proposed, both for the decomposition into a lower-dimensional representation and for the predictive model that relates this representation to the simulation inputs. While the dimension reduction is often obtained using PCA, which is a linear method, non-linear approaches developed in data mining [23], have also been used, despite their greater complexity. These include locally-linear PCA, kernel PCA, and deep neural networks [2, 39, 4, 10]. For the predictive models, a range of interpolation techniques have been used, including radial basis function regression, Gaussian processes, and deep neural nets [41, 38, 5].

Several unique aspects of our problem make it impractical to directly apply these ideas:

- **The very large size of each snapshot:** Many of the problems considered in the literature for building spatio-temporal surrogates have data at spatial points that number in the thousands, or tens of thousands, while a few have hundreds of thousands of grid points. For example, among the larger-sized data sets, one test problem in fluid flow considered by Rajaram et al. [29] had 1047 snapshots, each with 41,796 grid points and the other had 1001 snapshots, each with 450,000 grid points. Both problems were static (no time dependence), with a low-rank, reduced-dimensional space. A problem with more complex geometry considered by Bērziņš et al. [5] had a structured grid with 2459 nodes and two data sets, one with 100 simulations and the other with 400 simulations, where each simulation was run for 100 time steps. In addition, 50 simulations each were generated for validation and testing. In contrast, each snapshot in our data set has over 2 million grid points, which makes it challenging to both manage and process the data, requiring the algorithms to be modified suitably.

- **The small number of simulations and time steps:** Spatio-temporal models are typically built using a large number of simulations, numbering in the hundreds or thousands, as noted earlier. We found one example of a compute-intensive simulation [40] where just thirty simulations were run, each for 1000 time steps. While our constraints on resources also limit us to a similar number of simulations, we run each simulation for a much smaller number of time steps, as indicated in Table 1.

- **The variable size of the domain:** Most problems considered in spatio-temporal modeling are formulated on a fixed domain, with grid points at the same fixed locations at all time steps across all simulations. There are a few exceptions; the work by Yeh et al. [40] considers a problem where the input parameters control the geometry of the domain, while the problem in Bērziņš et al. [5] had a moving grid with a fixed number of grid points. As we show in Section 4.2, substantial pre-processing of our data is required to bring all snapshots into a common grid before we can build the surrogate.

Our contributions in this report are three-fold: First, we address the issues above and show how we process the large number of grid points per snapshot on domains that vary with each simulation. Second, we propose ways to determine the parameters for the algorithms used in building the surrogates. Finally, we investigate the accuracy of surrogates we might expect when the number of simulations and the number of time steps at which we run each simulation are both small. We show how we can use the small number of simulations

to identify the range of input parameters and to create a data set with a sufficient diversity of outputs. We also consider simple ways in which we can improve the quality of the surrogate while keeping the number of simulations small.

4   Solution approach

Our approach to building accurate spatio-temporal surrogates for jet-HE interaction problem is composed of multiple steps which we describe in detail in the following sections. We started by carefully generating sample points in the three-dimensional input parameter space (Section 4.1). We then pre-processed the output from each simulation so that all snapshots from all simulations had values defined at the same set of grid points, with the plate locations aligned at the initial time step, $t = 0$ (Section 4.2). This allowed us to represent the data in the form of a matrix, whose 1604 columns represented all the snapshots from the 45 simulations, and each row represented the value of the variable at a specific (x, y) grid point location. This data in the form of the snapshot matrix was used to build the spatio-temporal model (Sections 4.3 and 4.4) and the accuracy of the surrogate evaluated on new simulations (Section 5).

4.1   Sampling the input parameter space

One of the challenges in our problem is that simulating the interaction of the jet with the HE is compute intensive, limiting the number of sample points at which we can run the simulations. Therefore, the location of these sample points has to be selected carefully, which is challenging as we do not know a priori the outcome of running the simulation at any sample point or the range of input parameters we should use in generating the samples.

Therefore, we generated the sample points incrementally in the three-dimensional input space using a modified version of the best candidate algorithm [27, 15] that selects samples randomly, but far apart from each other. We first generated a small set of 16 samples using a range of [5.0, 20.0]cm for HE-length, [0.125, 0.25]cm for the jet radius, and [0.6, 0.95]cm/μs for the jet tip velocity. We then excluded four sample points with HE-length greater than 16.2cm, as this length was too large for the jet to even reach the plate at late time. Next, restricting HE-length to be in the range [5.0, 16.2]cm, we ran 12 new samples. Of the resulting 24 samples, we found that we could shrink the range of HE-length further to [5.0, 15.0]cm and also exclude samples in the lower right corner of the (HE-length - jet-tip-velocity) plot where the jet tip velocity was too low for the plate to break. This left us with data from 20 simulations, from an initial set of 28 simulations.

Having identified the range of values for generating simulations, we added 25 new samples in this region, for a total of 45 samples. As the best-candidate method is a progressive sampling algorithm, it allows us to add samples incrementally, while preserving the random and far-from-each-other property of the samples.

Our data set, shown in Figure 3, indicates that at high jet tip velocity, but low HE-length, the plate breaks, while at low jet tip velocity and high HE-length, the jet does not penetrate the plate. This latter region is sparsely sampled as we are interested mainly in cases where the plate breaks. The class label (break, no break, or almost break) was assigned by examining the outputs at the last two time steps in each simulation.

It is clear that our data set is unbalanced as we have 9 samples where the plate does not break, 31 where the plate breaks, and 5 that are almost break. Generating an appropriate data set for a problem like ours is challenging as we can run only a limited number of simulations. However, the boundary between the classes is poorly defined and we do not know the range of inputs that will give us sample points with the desired outcome. We erred on the side of having more break cases as these were of greater interest. The outputs for the no-break cases tended to be very similar, and we expected that a small number of such cases would suffice. Admittedly, as shown in Section 5, our choice of sample points affects the accuracy of the spatio-temporal surrogates built using the data set.

These 45 simulations generate a total of 1604 snapshots that vary in the number of grid points as indicated for the four examples in Table 1. The simulation at the extreme corner of the input space, with HE-length, jet tip velocity, and jet radius equal to 5.0cm, 0.950cm/μs, and 0.125cm, respectively, is referred to as the
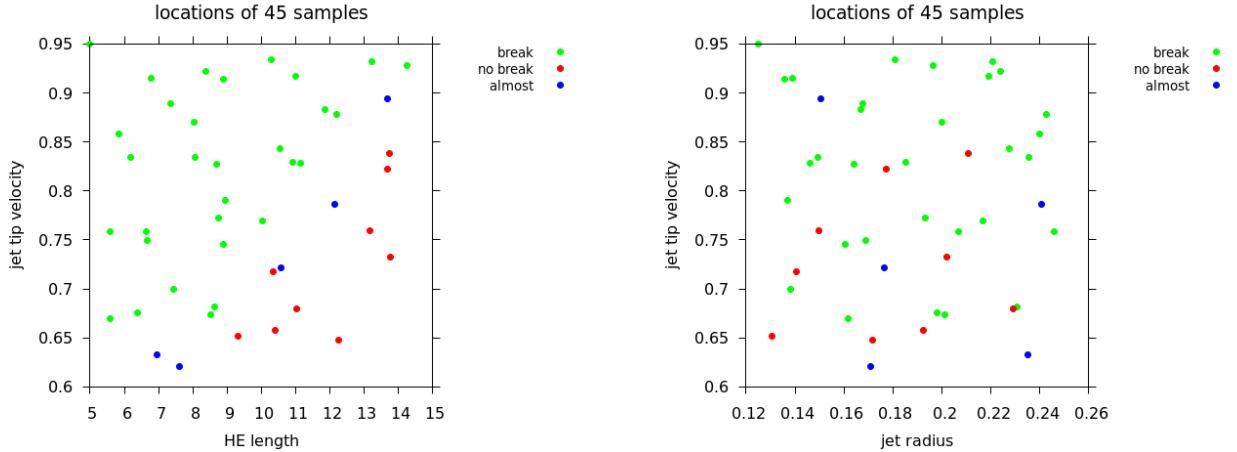
Figure 3: The 45 samples in the space of the three input parameters for the simulation, labeled by the state of the plate at the last time step of the simulation. The simulation at the extreme corner of the input space, with HE-length, jet tip velocity, and jet radius equal to 5.0cm, 0.950cm/μs, and 0.125cm, respectively, is referred to as the *baseline* simulation. It has the smallest number of time steps, with 29 snapshots.

baseline simulation. It has the smallest number of time steps, with 29 snapshots.

4.2   Pre-processing the data

To create a spatio-temporal surrogate for our data set, we first need to create a *snapshot matrix* for each of the three output variables, mass, x-momentum, and y-momentum. This snapshot matrix, $\mathbf{X} \in \mathbb{R}^{D \times N}$, is just a collection of the snapshots

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^D$, $N = 1604$ is the number of snapshots, and D is the number of grid points in a snapshot. However, a number of issues have to be addressed before we can generate the snapshot matrix where each snapshot has the same number of grid points at the same (x, y) coordinates. These issues, and our solution approach, are discussed in detail in the companion report [17] and described briefly below.

The data generated for each simulation, regardless of the size of the domain, are available in 360 files in HDF5 format [34] for each time step, as shown in Figure 4 for two of the four example sub-domains. Each HDF5 file includes five variables — x and y coordinates, mass, x-momentum, and y-momentum. Within each file, the variables are in natural ordering, that is, ordered by increasing values of the y-coordinate, and for a fixed y-coordinate, ordered by increasing values of the x-coordinate. All simulations are on a regular grid with $\Delta x = \Delta y = 0.0125$cm.

However, there are several differences in the data across the simulations that preclude just appending the snapshots to create the matrix $\mathbf{X}$. Figure 2 shows that each simulation has a different domain size, with the same range of values in y, but different ranges in x. In addition, while all simulations write the output to 360 files, their sizes are different as shown in Figure 4. We also observe that the plate, which is a key structure in the problem, is at different locations in the domain due to varying values of HE length in the simulations. A closer look at the (x, y) coordinates of the grid points indicated that the values of Δx and Δy are not exactly 0.0125cm across simulations, and while the coordinates in y, which has a fixed range of values, are identical across simulations, this is not the case for the x coordinates.

To address these differences across simulations, we first aligned the domain for each simulation so that the origin was at the lower right corner, which automatically aligned the plate at time $t = 0$ across all simulations. Next, we cropped the left end, removing data with the shifted x coordinate outside the range [-32, 0.0], which corresponds to the smallest HE length of 5.0cm. Now, all snapshots have data on the same domain, though

9
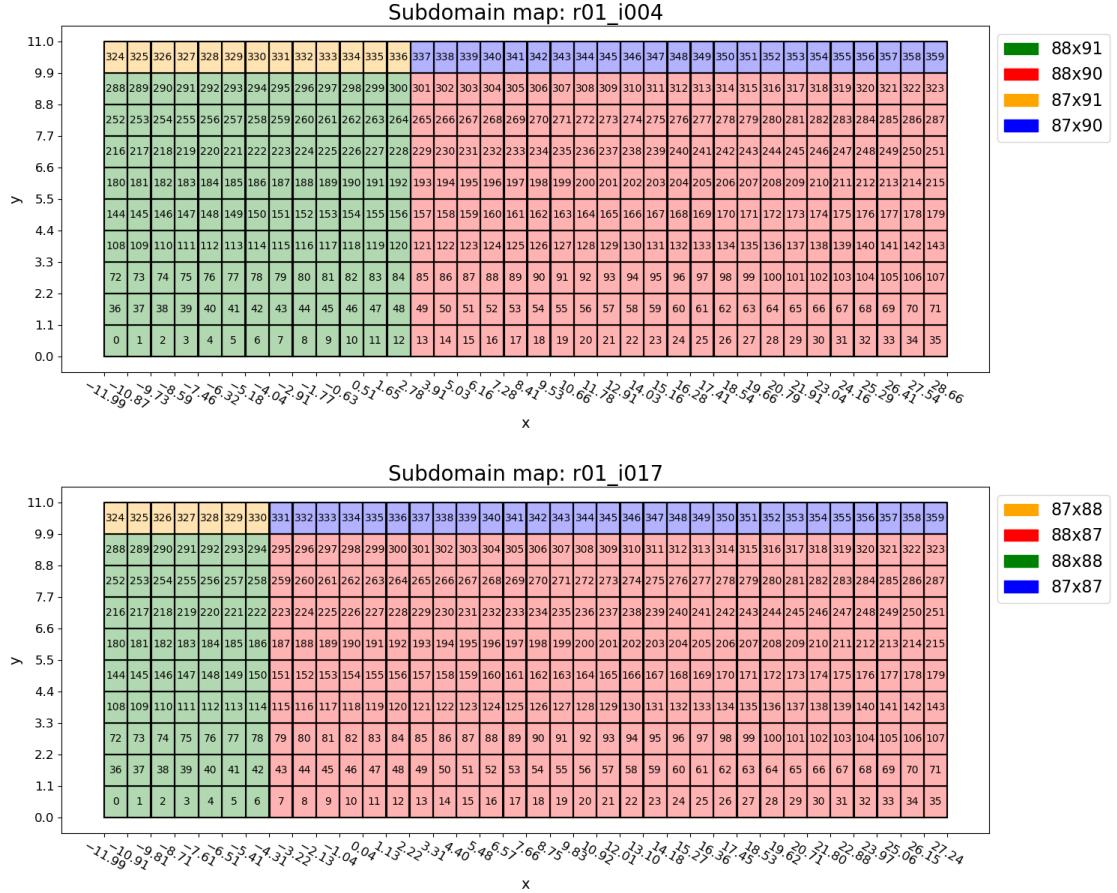
Figure 4: The distribution of data in 360 sub-domains for the first two example simulations in Table 1. Top: key r01_i004 and bottom: key r01_i017. At each time step, the data in each sub-domain is available in a separate HDF5 file. Note that the sizes of the sub-domains within a simulation vary and these sizes are different across simulations. All time steps in a simulation have the same distribution of the data.

not at the same (x, y) coordinates. To accomplish this, we re-mapped the data to a common grid, using a simple 1-nearest neighbor algorithm.

To make further processing of the large snapshot matrix feasible, we generated it in a block form

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_{b1} \\ \mathbf{X}_{b2} \\ \dots \\ \mathbf{X}_{bk} \end{bmatrix} \tag{2}$$

by using a block form for the common grid that was used to remap the data. Each block was written to a separate file and contained all grid points in a specific non-overlapping range of y values, stored in natural order. Thus, concatenating the blocks in the order of their y values, would result in a single snapshot-matrix file for each variable, with the grid points stored in natural order in the rows of the matrix.

At the end of the pre-processing step, we have each of the three output variables, in a separate snapshot matrix, where each matrix has 1604 columns and 2,180,799 rows. Each matrix is split by rows into 22 blocks, each block stored in a separate file. Figure 5 shows the mass variable for the first and last snapshot, for each of our four example simulations, after the raw output data have been aligned, cropped, and remapped. The corresponding images for x-momentum and y-momentum are shown in Figures 12 and 14 in Appendix A and B, respectively.
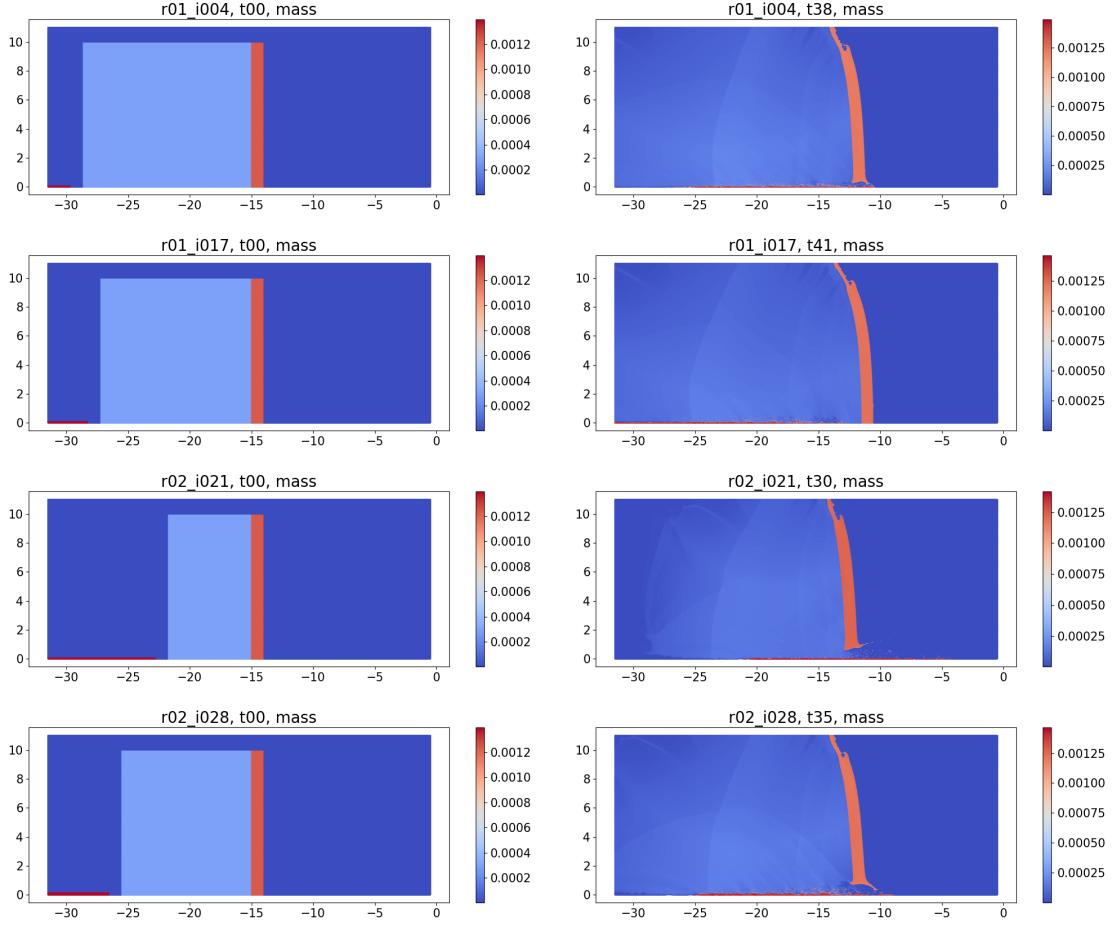
Figure 5: The variable *mass*, after the snapshots have been aligned, cropped, and remapped. Left column shows the first time step and right column shows the last time step for the four example simulations in Table 1. From top to bottom, simulations with keys r01_i004, r01_i017, r02_i021, r02_i028, respectively. The color bars are different between simulations and across time steps.

Next, to illustrate the evolution of the simulations over time, we show the data, after the original output has been pre-processed, at multiple time steps in two example simulations, key r01_i017 and key r02_i028, corresponding to no-break and break cases, respectively. Figure 6 shows the mass variable evolving with time in these two simulations. The corresponding images for x-momentum and y-momentum are shown in Figures 13 and 15 in Appendix A and B, respectively.

The pre-processing of the data described in this section is similar to that performed in face recognition, where the face images are all processed so they are the same size, with the face centered in the image and major features such the eyes and nose aligned across images. The difference in our data is that unlike a face, which is stationary, the snapshots show the motion of the HE and the plate with time. Therefore, even though we aligned the plate at time $t = 0$ in all simulations, there is a loss of alignment of the plate location as the simulations progress, which will influence the accuracy of any surrogate built from the data.

4.3   Surrogate using a linear transformation

After pre-processing the simulation output and converting it into a snapshot matrix, $\mathbf{X}$, in a block form as shown in Equation 2, we build the spatio-temporal surrogate using the traditional approach outlined in Section 3. We start by linearly transforming the snapshot matrix $\mathbf{X}$ using a singular value decomposition

11
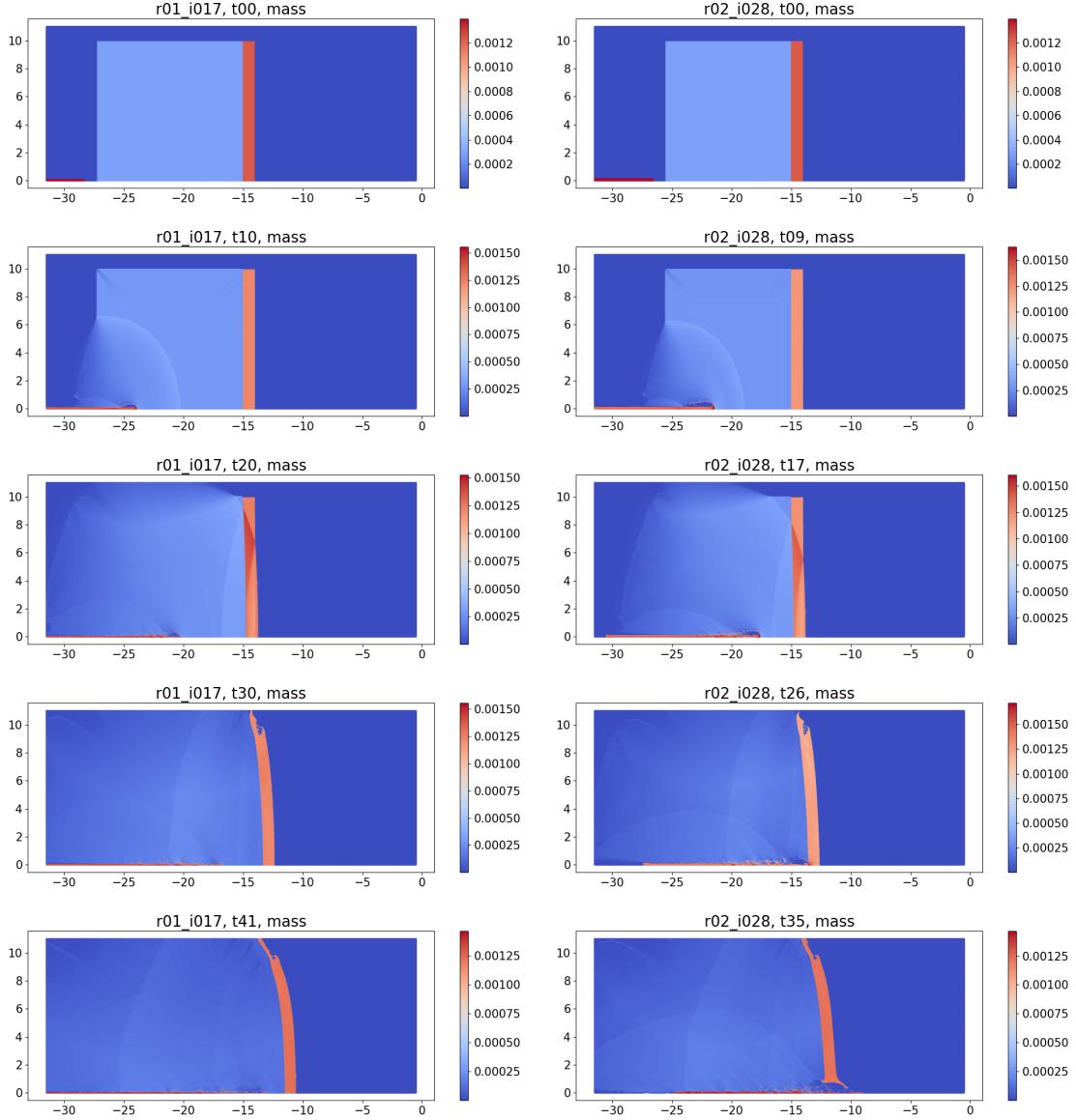
Figure 6: The variable *mass*, after the snapshots have been aligned, cropped, and remapped, at different time steps in two simulations showing the evolution of the data over time. Left: key r01_i017 (no break case) at time steps t0, t10, t20, t30, and t41. Right: key r02_i028 (break case) at time steps t0, t09, t17, t26, and t35. The color bars are different between simulations and across time steps.

(SVD) [26]

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \tag{3}$$

where

$$\mathbf{X} \in \mathbb{R}^{D \times N} \quad, \quad \mathbf{U} \in \mathbb{R}^{D \times D} \quad, \quad \boldsymbol{\Sigma} \in \mathbb{R}^{D \times N} \quad, \quad \text{and} \quad \mathbf{V} \in \mathbb{R}^{N \times N}.$$

Here, $\boldsymbol{\Sigma}$ is a diagonal matrix with non-zero diagonal elements σ_{ii}, referred to as the singular values of the matrix $\mathbf{X}$. These singular values are typically ordered in descending order, with the rows and the columns of the $\mathbf{U}$ and $\mathbf{V}$ matrices ordered correspondingly. Since $D >> N$ in our problem, the top N rows of $\boldsymbol{\Sigma}$ will have non-zero diagonal elements, assuming rank($\mathbf{X}$) = N. If the rank, k, is less than N, then only the first k diagonal elements will be non-zero and the rest will be zero. The columns of the orthonormal matrices $\mathbf{U}$

and $\mathbf{V}$ are referred to as the left and right singular vectors of $\mathbf{X}$. The matrix

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N] \quad \text{where} \quad \mathbf{u}_i \in \mathbb{R}^D \tag{4}$$

is also the matrix of orthonormal eigenvectors of $\mathbf{X}\mathbf{X}^T$, and the singular values σ_i are the square-root of the eigenvalues of $\mathbf{X}\mathbf{X}^T$ (or $\mathbf{X}^T\mathbf{X}$). We refer to the vectors $\mathbf{u}_i$ as "*eigen-snapshots*", akin to the "eigenfaces" used in face recognition [35].

The $\mathbf{u}_i$ also form a basis in $\mathbb{R}^D$ for the data, so each snapshot $\mathbf{x}_i$ can be written as a linear combination of the $\mathbf{u}_k$ as follows

$$\mathbf{x}_i = \sum_{k=1}^{N} w_{ki}\mathbf{u}_k \tag{5}$$

where the weight, w_{ki}, of the k-th basis for the i-th snapshot is given as

$$w_{ki} = \mathbf{u}_k^T \mathbf{x}_i \quad \text{for} \quad k = 1, \ldots, N . \tag{6}$$

These weights are just the projection of the i-th snapshot onto each of the columns of the $\mathbf{U}$ matrix. Equation 5 shows how the original snapshot can be reconstructed using the weights and the basis vectors. A reasonable approximation, $\tilde{\mathbf{x}}_i$, to a snapshot $\mathbf{x}_i$ can then be obtained by using the $\mathbf{u}_i$ corresponding to the $n, n < N$, larger singular values:

$$\tilde{\mathbf{x}}_i = \sum_{k=1}^{n} w_{ki}\mathbf{u}_k. \tag{7}$$

Thus, if we can generate the SVD for $\mathbf{X}$ as in Equation 3, then Equations 6 and 7 enable us to generate an approximation, $\tilde{\mathbf{x}}_i$, to the original snapshot, $\mathbf{x}_i$. These equations can be interpreted as generating a reduced representation in two ways - first, by creating an approximation that uses a smaller number of weights and basis functions, ignoring the other weights and basis functions, and second, by characterizing a snapshot $\mathbf{x}_i$ in terms of just its weights

$$\mathbf{w}_i = [w_{1i}, w_{2i}, \ldots, w_{Ni}], \tag{8}$$

which can be combined with the basis set $(\mathbf{u}_k, k = 1, \ldots, N)$ to reconstruct the snapshots exactly.

This representation of the i-th snapshot, in terms of its weights, $w_{ki}, k = 1, \ldots, N$, can be related to another representation of the snapshot in terms of its simulation input parameters (h_i, v_i, r_i) and time step, t_i, through a predictive model

$$w_{ki} \approx f_k(h_i, v_i, r_i, t_i), \text{ where } k = 1, \ldots, N \text{ and } i = 1, \ldots, N. \tag{9}$$

Here h_i, v_i, and r_i are the HE length, jet tip velocity, and jet radius for the i-th snapshot. Note that a separate model is created for each weight index, that is, the first or most important weight is predicted by one model, the second most important weight by another, and so on, resulting in the n (or N) models required to generate the snapshots approximately (or exactly). Then, to reconstruct approximately the two-dimensional output, or snapshot at a new set of input parameters and time step, we can first use the predictive model for each of the n weights to predict the weight values at this new input and time step, and apply Equation 7.

The spatio-temporal surrogate is thus composed of the basis functions obtained using the SVD and the predictive models created using a training set that relates the simulation inputs and time step to the weights associated with the basis functions. There are several options available to calculate the SVD and to generate the predictive models; we next describe the options we selected for use in our problem.

We considered two ways to generate the SVD of $\mathbf{X}$ and obtain the matrix $\mathbf{U}$ and the singular values σ_i:

- In the first approach based on the **normal equations**, we form the matrix $\mathbf{X}^T\mathbf{X}$, of size $N \times N$, and obtain its eigen-decomposition. The singular value σ_i of $\mathbf{X}$ is then the square-root of the eigenvalue λ_i of $\mathbf{X}^T\mathbf{X}$. The columns $\mathbf{u}_i \in \mathbb{R}^D$ of $\mathbf{U}$ are the corresponding orthonormal eigenvectors of $\mathbf{X}\mathbf{X}^T$. Since

we have calculated the eigenvectors, $\mathbf{v}_i \in \mathbb{R}^N$, of the much smaller matrix $\mathbf{X}^T\mathbf{X}$, we can obtain the $\mathbf{u}_i$ as

$$\mathbf{u}_i = \frac{1}{\sqrt{\lambda_i}}\mathbf{X}\mathbf{v}_i \quad . \tag{10}$$

The division by $\sqrt{\lambda_i}$, which is the length of the vector $\mathbf{X}\mathbf{v}_i$, results in orthonormal vectors $\mathbf{u}_i$.

This normal equations approach is very straight-forward and easily extends to the case when $\mathbf{X}$ is in block form (Equation 2) as we can form the matrix $\mathbf{X}^T\mathbf{X}$ by reading in a block at a time. However, it does suffer from floating point issues associated with normal equations [26].

- A more stable approach to SVD is the **QR decomposition** that does not involve forming the normal equations. Here, we first decompose the snapshot matrix as $\mathbf{X} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q} \in \mathbb{R}^{D \times D}$ has orthonormal columns and $\mathbf{R}$ is upper triangular, and then generate the SVD of the smaller matrix $\mathbf{R}$. Since $\mathbf{X}$ is a tall, skinny matrix, we use the thin/reduced version of the QR decomposition [37]:

$$\mathbf{X} = \mathbf{Q}\mathbf{R} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ 0 \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1 \tag{11}$$

where

$$\mathbf{X} \in \mathbb{R}^{D \times N} \quad , \quad \mathbf{Q} \in \mathbb{R}^{D \times D} \quad , \quad \mathbf{Q_1} \in \mathbb{R}^{D \times N} \quad , \quad \mathbf{R_1} \in \mathbb{R}^{N \times N}.$$

We next obtain the SVD of the much smaller matrix $\mathbf{R_1}$

$$\mathbf{R}_1 = \mathbf{U}_{R_1}\mathbf{\Sigma}_{R_1}\mathbf{V}_{R_1}^T \tag{12}$$

where $\mathbf{U}_{R_1} \in \mathbb{R}^{N \times N}$, $\mathbf{\Sigma}_{R_1} \in \mathbb{R}^{N \times N}$, and $\mathbf{V}_{R_1}^T \in \mathbb{R}^{N \times N}$, which gives

$$\mathbf{X} = \left(\mathbf{Q}_1\mathbf{U}_{R_1}\right)\mathbf{\Sigma}_{R_1}\mathbf{V}_{R_1}^T. \tag{13}$$

Thus, the singular values of $\mathbf{X}$ are just the singular values of $\mathbf{R}_1$ and the left singular vectors of $\mathbf{X}$. that is, the basis functions, are the columns of $\mathbf{Q}_1\mathbf{U}_{R_1}$.

To extend the QR decomposition to matrices in the block form, we used the work of Constantine and Gleich [8]; this implementation is less straight-forward than the normal equations.

We have several options for the n predictive models to predict the weights in the reduced representation of the data. In our work, we use a machine learning model, specifically, a Gaussian process model [30], as it is one of the models that is accurate for small data sets [16]. It also provides an estimate of the uncertainty on the prediction, which gave us an opportunity to investigate whether the uncertainty could be used to understand and explain the results, and to determine the number of weights, n, to use in the reduced representation. The GP is an expensive model to create, especially when we use automatic relevance determination [30], where the hyper-parameters include the weights on each input. However, in our problem, we can create and apply the n models in parallel, which reduces the turnaround time for creating the models.

4.4 Surrogate using a locally-linear transformation

The singular value decomposition, described in Section 4.3 is a linear decomposition and if the data do not lie on a linear manifold, the number of weights required for a reduced representation may be quite large. This can create problems with accurate prediction of the weights when we have a small number of simulations.

A simple approach to introducing nonlinearity in the decomposition is to use a locally-linear decomposition, that is, cluster the snapshots by similarity and then apply the linear, SVD-based approach to each separate cluster [20, 19, 12]. Then, to predict the two-dimensional output at a new point in the simulation input space and time step, we identify which cluster the new point belongs to, and use the predictive model created for that cluster.

A challenge in clustering our data is the extreme high-dimensionality of the snapshots as each snapshot has over 2.2M grid points. As described in the companion report [17], we investigated multiple ways to

address this problem. For iterative clustering methods, such as k-means, which require multiple passes through the data, processing large files in each iteration can be expensive. We considered two ways to avoid this processing: in the first approach, we reduced the dimension prior to clustering by using random projections [9, 24] and, in the second approach, we clustered a different, but equivalent, representation of the each snapshot, such as the weights obtained after SVD (Equation 6). We also investigated a non-iterative algorithm, hierarchical clustering [13], which requires the pairwise distances between the snapshots. By calculating the distance matrix once, we could experiment with different options for the method. We found that hierarchical clustering with Ward linkage gave remarkably similar results to k-means clustering. We chose the results with k-means and random projections as we could exploit the randomness of both the random projections and the initial choice of cluster centroids in k-means to identify the number of clusters.

A locally-linear transformation that combines clustering of snapshots, following by an SVD on each cluster, is not the only option for non-linearly transforming our data. Other options include kernel PCA [39], which requires the solution of an ill-posed inverse problem to reconstruct the two-dimensional output at a new point in the simulation input space, and a neural-network based auto-encoder, which, given the high-dimensionality of the snapshots, will be a challenge to implement, even with a small number of hidden layers [19]. In this work, we selected the locally-linear transformation as it is a simple method and allowed the re-use of software developed for the linear transformation.

5   Experiments, results, and discussion

We next present the results of our experiments to test the accuracy of our two spatio-temporal surrogates, one using a linear decomposition and the other using a locally-linear decomposition. Recall that the approach in both cases is similar: we start with a snapshot matrix, generate an SVD for it, and calculate the eigen-snapshots and the corresponding weights that can be used to perfectly reconstruct each snapshot. Next, we build Gaussian process models that can predict each of the initial, important weights based on the simulation inputs and time step. Then, to reconstruct an approximation to a new snapshot that is identified by its simulation input parameters and time step, we first predict the weights and then combine the weights with the eigen-snapshots. The two surrogates differ in the snapshot matrix that is used - the linear surrogate uses a matrix composed of all snapshots, while for the locally-linear surrogate, we cluster the snapshots, grouping similar snapshots together, and then build a linear surrogate for each cluster separately.

Our eventual goal in building these spatio-temporal surrogates is to predict what happens to the jet and the plate at late time, specifically, does the jet go through the plate, what is the velocity of the jet tip when it goes through the plate, and what is the location of the plate at late time. As the simulations are expensive, we want to build accurate surrogates using only a small number of data points in the simulation input space. As stated earlier, in this initial study, we qualitatively evaluate the surrogates to understand to what extent we can achieve our goals and to identify simple ways in which we can improve the accuracy of the surrogates without substantially increasing the number of simulations required.

In our work, given the questions we want to address regarding the status of the plate and the jet at late time, we focus on the mass and x-momentum variables. As shown in Figures 14 and 15, the y-momentum variable does not clearly define either the plate or the jet, and is therefore less useful in our analysis.

In the following sections, we describe the test data we use to evaluate the accuracy of reconstruction of new snapshots and discuss how we determine various parameters used in our approach, including the number of weights to use in the reconstruction, the number of clusters to use for the locally-linear surrogate, and the identification the cluster to which the new data point belongs. We also discuss ways to evaluate the reconstruction quality for the test snapshots.

5.1   Test simulations

To test our ideas and evaluate the accuracy of the linear and locally-linear surrogates created using the 45 simulations shown in Figure 3, we identified seven data points in this simulation input parameter space.

Simulation key	jet radius (cm)	HE-length (cm)	jet tip velocity (cm/μsec)	# time steps	outcome
r03_i008	0.221386	12.5439	0.725508	40	no break
r03_i018	0.153774	12.8638	0.830377	38	almost break
r03_i021	0.1647	11.7005	0.760524	38	almost break
r03_i023	0.197432	9.0411	0.613552	37	no break
r03_i026	0.219371	5.86578	0.912855	29	break
r03_i037	0.246562	9.47074	0.867685	33	break
r03_i050	0.211678	8.08681	0.794035	33	break

Table 2: Input parameters for the seven test simulations used to evaluate the quality of the spatio-temporal surrogates.
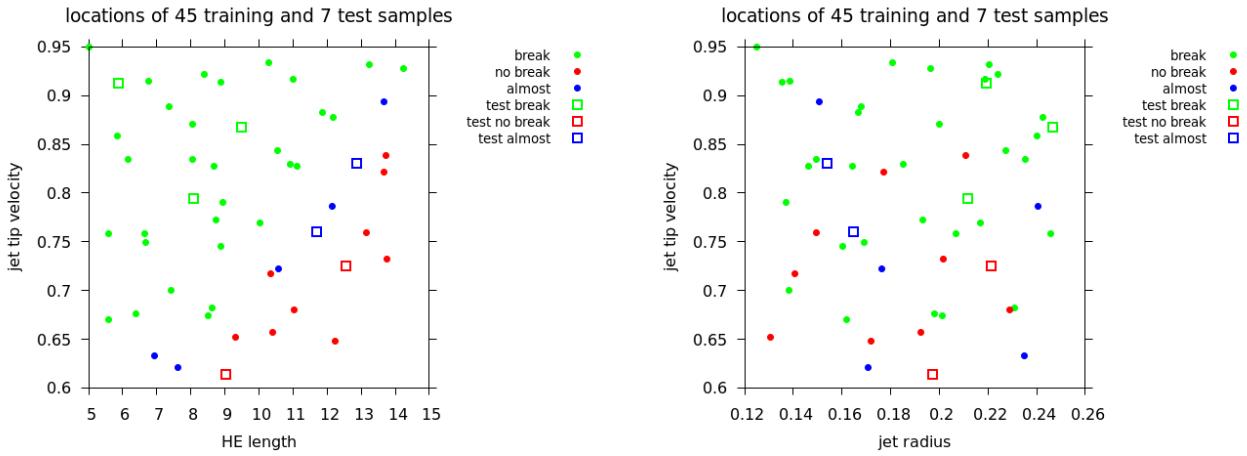
Figure 7: The 7 test simulations, along with the 45 training samples, in the space of the three input parameters, labeled by the state of the plate at the last time step of the simulation. Note that several of the test points are either near the boundary of the rectangular domain, or near the boundary of the region identified by the data points.

These points are listed in Table 2 and their locations are shown in Figure 7. The parameters for these data points were selected to be different from each other, so we could evaluate the quality of the predictions not only at different locations in the input parameter space, but also at different locations in the break, no-break, and almost-break space. Three of the seven points are clearly in the region where the plate breaks, with two points each in the almost-break region and the no-break regions. The latter four points were more difficult to identify as the no-break region is less well defined in the (HE-length - jet-tip-velocity) space, and the no-break points are near the boundary of the sampled region. Their locations make it likely that the output predictions at these four data points will be less accurate.

To enable us to evaluate the outputs predicted by the surrogates by comparing with the actual outputs, we ran the simulation at these seven points in the input space and identified the status of the plate at the last time step. In practice, this last time step would be identified as described in Section 2.

5.2   Generating the SVD

There are several issues we need to address related to the implementation of the SVD. As explained in Section 4.3, we considered two implementations of the SVD algorithm - one based on the normal equations and the other on a QR decomposition. In early experiments, using a snapshot matrix with data from fewer than 45 simulations, we found that when select snapshots were reconstructed using all the weights, the error

was greater when we used the normal equations formulation. This was expected behavior given the floating point issues associated with the normal equations. Therefore, all results in this report were generated using the QR decomposition.

Another issue that is the subject of much confusion is whether the snapshot matrix should be "centered", that is, the mean snapshot subtracted from each snapshot, before calculation of the SVD. As observed by Bērzinš et al. [5], there are adherents on both sides of the issue. Jolliffe and Cadima [7, 14] have discussed the topic at length, providing examples of where centering may or may not be meaningful. Some authors [1], taking a pragmatic approach, have evaluated the results both ways, and selected the option that best met their needs. For our data set, the mean of the snapshot matrix with all 1604 snapshots is dominated by the plate at early time; this is because in most snapshots, the plate, which is aligned across all simulations at $t = 0$, has barely moved from the initial location. Therefore, when the mean snapshot is subtracted from the snapshots at late time (the ones of most interest to us), the initial position of the plate appears clearly. As a result, more eigen-snapshots are required to account for, say, 90% of the variation in the data. In addition, we found that the substantial visual change in the late time snapshots after centering made it difficult to determine how many weights to use in the reconstruction.

5.3   Clustering the snapshots

Clustering the 1604 snapshots for each of the three variables proved to be challenging for several reasons. In Section 4.4, we discussed how we addressed the issue of the extremely high-dimensionality of each snapshot. We also wanted to understand if there was an inherent clustering in the collection of snapshots. A careful analysis of the snapshots, as shown in Figures 5 and 6 for mass, and Figures 12 and 13 for x-momentum, indicated that the first and last snapshots in a simulation are quite different, suggesting that multiple clusters exist in the data. However, clustering the snapshots would result in some neighboring snapshots, which are very similar, assigned to two different clusters, which suggested that the clusters are not well separated. This made it difficult to identify the number of clusters and to evaluate the results of any clustering algorithms.

A detailed discussion on how we addressed these issues is given in the companion report [17]. Figure 8 shows the clustering results we use for the mass and x-momentum variables. These results were obtained using the k-means clustering algorithm, combined with random projections to reduce the dimensionality of the snapshot matrix. Based on these results, for our problem, it is relatively simple to identify the cluster for each of the snapshots at which we want to predict the 2-D output. Our interest is in the late time cluster, which is cluster 2 for the mass variable and cluster 0 for the x-momentum variable. Using these clusters, we can predict the output at both the last time step and the time step that is 2 prior to the last time step; we refer to these time steps as tlast and (tlast-2). Our reason for predicting two late-time snapshots in each of the seven test simulations will become clear in Section 5.6.

5.4   Determining the number of singular values to keep

One of the key issues in reconstructing the simulation outputs at a new data point is the determination of n, which is the number of weights and basis vectors to use in building the approximation. Ideally, we want a good approximation to a snapshot using a small value for n. Let $\mathbf{X}_n$ be the matrix composed of snapshots reconstructed using only the top n weights. For our problem, with a tall, skinny $\mathbf{X}$, this error in the reconstruction is given by

$$\|\mathbf{X} - \mathbf{X}_n\|_F^2 = \sum_{i=n+1}^{N} \sigma_{ii}^2 \tag{14}$$

that is, the sum of squares of the singular values that were excluded in the reconstruction. If these singular values are small, the error in the approximation is small. The cumulative percentage variation explained by the first n singular vectors is

$$\frac{\sum_{ii=1}^{n} \sigma_{ii}^2}{\sum_{ii=1}^{N} \sigma_{ii}^2} * 100.0 \ . \tag{15}$$

17
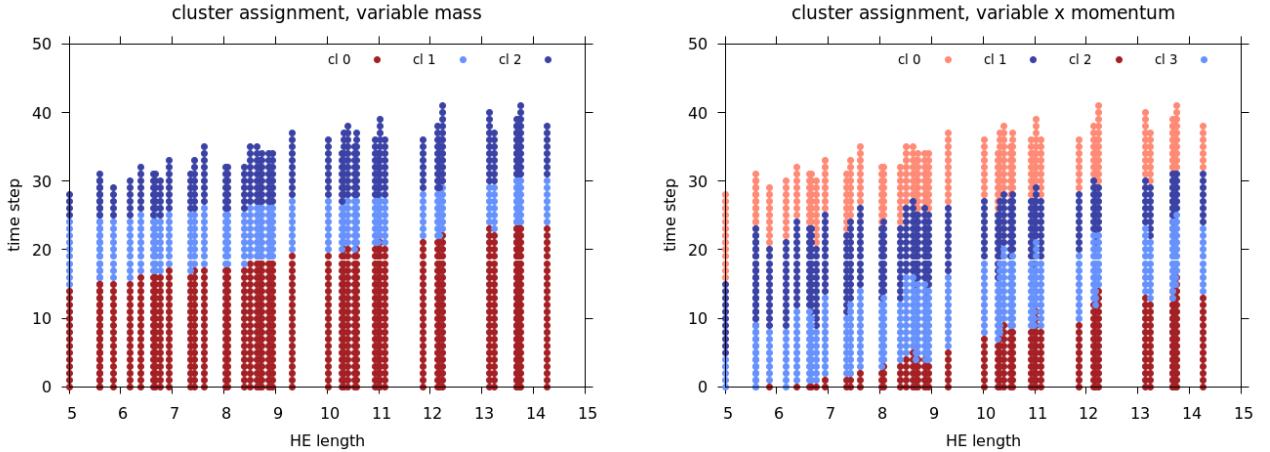
Figure 8: The cluster assignment, for the mass variable (3 clusters) on the left, and the x-momentum variable (4 clusters) on the right. The plots show the times steps in the simulations plot along with the HE length. For the mass variable, the cluster sizes for clusters 0, 1, and 2, are 889, 351, and 364 snapshots, respectively. For the x-momentum variable, the cluster sizes for clusters 0, 1, 2, and 3 are 397, 454, 305, and 448 snapshots, respectively.

Thus, one way to determine n is to use a fixed percentage variation, say 90% or 95%, that we would like explained by the reconstructed data, and identify the n associated with it. This is the most popular approach in building spatio-temporal surrogates, where, having identified n, predictive models are created for each of these n weights and then used to predict the weights at a new data point.

Data set	# snapshots	90%	95%
mass, unclustered	1604	6	17
mass, cluster 2	364	7	28
x-momentum, cluster 0	397	40	130

Table 3: Number of weights required to reconstruct the snapshot matrix to account for 90% and 95% variation in the data for the mass variable (clustered and unclustered) and the x-momentum variable (clustered).

Table 3 lists the values of n for our snapshot matrices. As the values of n can be quite large, and we have a small number of simulations, we wanted to confirm the quality of the n models with a leave-one-out approach prior to their use. We created each model with all but one of the snapshots, and used the model to predict the weight for the snapshot left out. For a good model, the plot of the predicted vs. actual weights should give points close to the $y = x$ line. However, we found that the quality of the models tended to deteriorate quite quickly as the index of the weight increased. Figure 9 shows a sample of these predictions for the mass variable, with all 1604 snapshots and with the 364 snapshots in the last cluster. We found that the most important weights usually predicted very well, but the scatter around the $y = x$ line increased with the weight index. Sometimes, many of the predicted values would be along the $y = x$ line, with some relatively large outliers. In other cases, most of the predictions were poor. Interestingly, some weights at higher indices had better predictions than the weights at lower indices.

We suspect that this poor prediction of some weights is due to the small size of our data set, as we have just 45 simulations. Bērziņš et al. [5] observed that for one of their problems, increasing the data set from 100 to 400 simulations did not affect the linear transformation, but significantly improved the accuracy of the models created to predict the weights.

The results in Figure 9 indicated that we could not use the quality of the actual vs. predicted weight for the training data set to determine the number of weights for reconstruction. Even when the overall quality of predictions on the training data are poor, a weight prediction at a new data point could be accurate, and

18
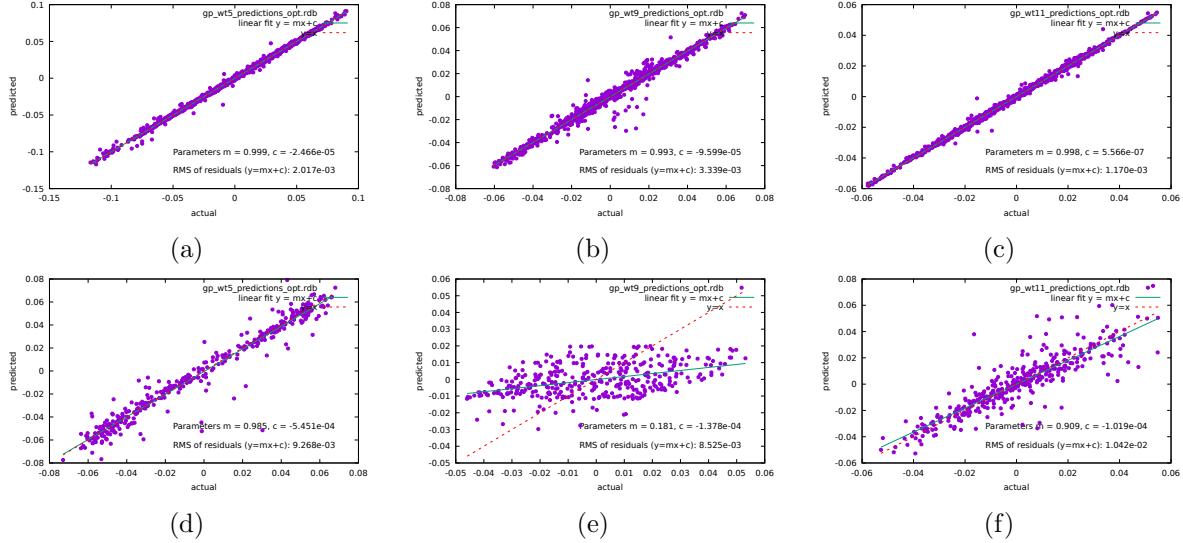
Figure 9: Evaluating the prediction of the weights by the Gaussian process models using a plot of actual vs. predicted values obtained using leave-one-out validation. Top row: mass, without clustering. Bottom: mass, cluster 2. From left to right, weight indices 5, 9, and 11 (the first weight has index 0). The red line is $y = x$ line, while the blue line is the best fit line to the data, whose slope and intercept is also included in each plot. The very early weights typically have little to no scatter as in (a), but the scatter increases with weight index, as in (b) and (d). The prediction may be incorrect for most snapshots, as in (e). It may also improve with increasing weight index, as in (c) and (f).

vice versa.

We therefore decided to estimate the number of weights based on the uncertainty in prediction obtained from the Gaussian process surrogates. We predicted the values of the first 50 weights and started by using the largest number of weights that all had small variance in the predictions. Since this number is relatively small for the mass variable, we also generated the reconstructions using a larger number of weights to understand the inaccuracies they would introduce into the reconstructed outputs.

5.5 Software and parallel implementation

Many of the steps in our approach to creating spatio-temporal surrogates can be parallelized to reduce the computational time. In this initial implementation, we exploited parallelism wherever it was possible to do so easily. For example, tasks in the pre-processing of the HDF5 files, such as reading the files, aligning and cropping the domains, and remapping each simulation to the common grid, could all be done in parallel across simulations using the Python sub-process function. For the SVD using QR decomposition on our block snapshot matrix, we implemented a serial version of the parallel algorithm by Constantine and Gleich [8]. For the random projections used prior to clustering, we used a sparse random matrix that was generated on the fly, reducing the storage required, and enabling better use of the cache on the computer system.

Where possible, we used pre-existing, highly optimized software from the double-precision BLAS [6] and LAPACK [3] libraries. These included DGEMV and DGEMM for matrix-vector and matrix-matrix multiply, respectively; GEQRF, GESVD, and ORGQR for the SVD using the QR decomposition; and DSYEVR for the eigenanalysis on the normal equations.

We also did not make any efforts to optimize the codes, for example by finding the optimal block size to use in storing the snapshot matrix or identifying ways in which consecutive steps in the processing could be merged for faster turnaround. Since our focus in this report is to understand what is possible when the number of simulations is small, we do not include compute times for various steps in our solution approach.

19

5.6 Reconstruction results

We next present the results of reconstruction of the tlast and (tlast-2) time steps for the seven test simulations using our two surrogates. For the mass variable, we show results for both linear and locally-linear transformations. As expected, the latter approach gave better results as the models to predict the weights were created using snapshots that were more similar to each other. Therefore, for the x-momentum variable, we considered only the locally-linear transformations.

To determine the number of weights to use in reconstruction, we generated the predictions and the uncertainties for the first 50 weights using the Gaussian process models. We expected, based on Table 3, that a maximum of 50 weights should suffice to obtain a reasonable reconstruction for the mass and x-momentum variables. Typically, the weights reduce in value with increasing index number, and later weights that are near zero can be dropped. However, it is possible that for a specific snapshot, a later weight is too large to ignore. We combined the information in Table 3 with the uncertainties in the weight predictions, to determine the exact number of weights to use in reconstruction as follows:

- **mass variable using the linear transformation:** Using the SVD results for all 1604 snapshots, we reconstructed the test snapshots with 8, 20, and 47 weights. We considered the first 8 weights as these weights were predicted with low uncertainty for all 14 test snapshots and Table 3 indicated that we would capture more than 90% of the variation in the training data. We considered 20 weights as 17 weights allowed us to account for atleast 95% variation in the training data, and adding weights 18 through 20, which were predicted with low variation for all 14 test snapshots, could only improve the results. Finally, we used 47 weights to understand the effects of adding more weights; we stopped at 47 as all test snapshots had higher uncertainty on prediction of weight 48. We observe that for many test snapshots, the six weights with indices ranging from 9 to 15, often had high uncertainty. We therefore expected reconstructions with more than 8 weights to be of poor quality, but we wanted to understand to what extent including weight predictions with larger uncertainties would influence the results.

- **mass variable using the locally-linear transformation:** Using the SVD results for the 364 snapshots in the late-time cluster, we generated the reconstructed test snapshots with 30 weights. The errors in the initial weights tended to be small, especially for the (tlast-2) time steps. The first large error typically occurred at weight 31, resulting in our choice of 30 weights for reconstruction.

- **x-momentum variable using the locally-linear transformation:** Table 3 indicated we needed 40 weights to capture 90% variation in the reconstructed snapshots. We used all 50 weights as the error in the prediction of weights at higher indices was often very small.

We next describe the reconstruction results using the two spatio-temporal surrogates. First, in Appendix C, we show the first ten eigen-snapshots for the mass variable generated with the linear and locally-linear transformations. The corresponding eigen-snapshots for the x-momentum variable, locally-linear transformation, are shown in Appendix E. In addition, for each of the 14 snapshots being reconstructed, we present the following detailed results:

- **the predictions obtained from the Gaussian process surrogate:** We plot the predictions for the first 50 weights, with the uncertainty in prediction shown as an error bar at 1 standard deviation.

- **the original snapshot, along with various reconstructed snapshots:** We focus on the region around the plate as we want to predict what happens to the plate at late time.

- **a y-lineout using the actual values in the reconstructed snapshots:** This plot of the variable values at $y = 6.0063$ and $-15.5 \leq x \leq -10.5$ (in our transformed coordinates) for the original and reconstructed snapshots gives a concise summary of the quality of reconstruction and makes the comparison more quantitative than comparing entire snapshots visually.

These detailed results, including weight predictions, reconstructed snapshots, and the y-lineouts for the linear and locally-linear surrogates for the mass variable at time steps tlast and (tlast-2) for all seven test

snapshots are shown in Appendix D. The corresponding results for the x-momentum with the locally-linear surrogate are shown in Appendix F. We repeat the y-lineout for the reconstruction of the 14 test snapshots for the mass variable in Figures 10 and 11 and use them to discuss the results and compare the different options used.
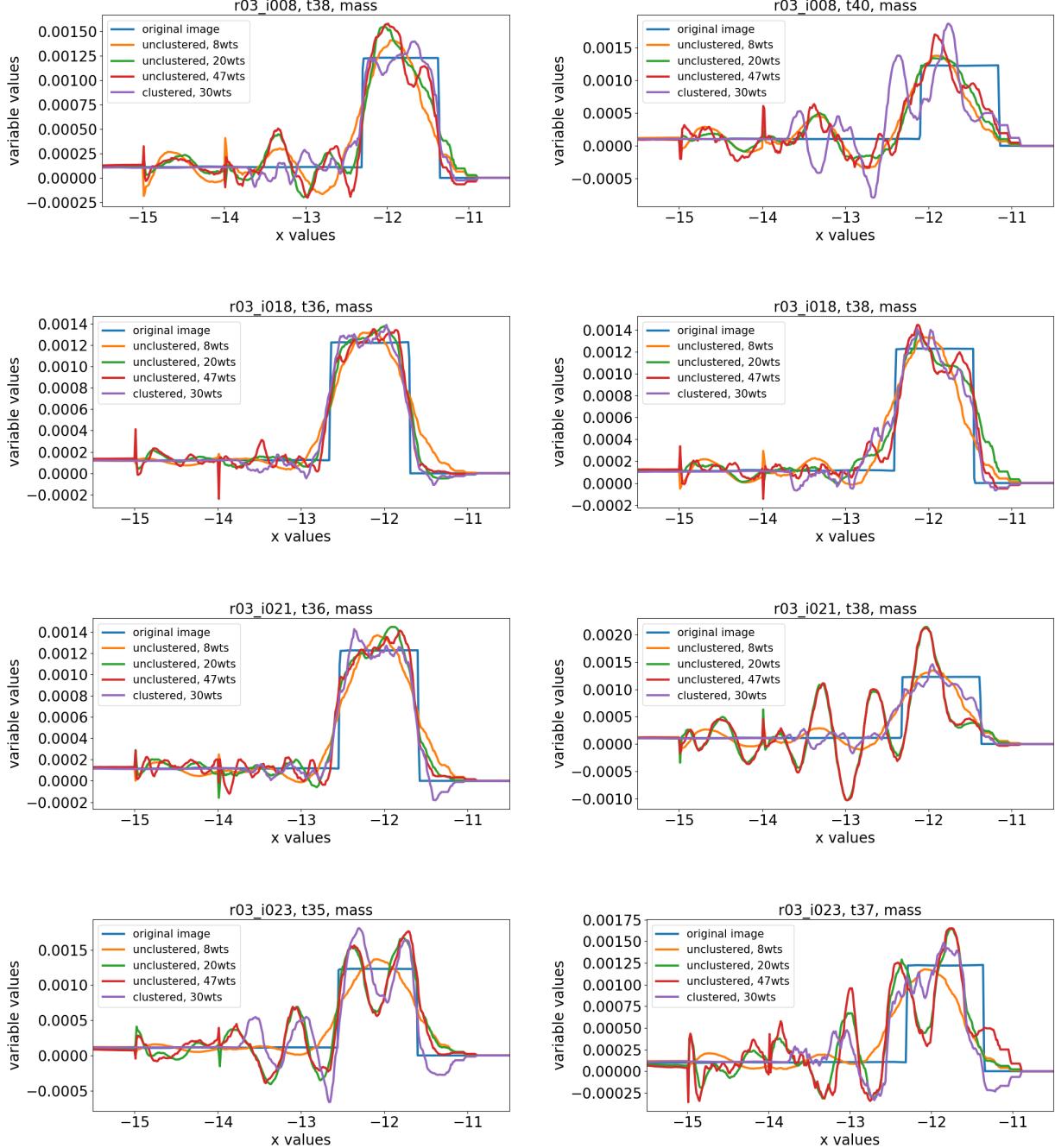

Figure 10: The y-lineouts for comparing the reconstruction of the *mass* variable both with and without clustering. From top to bottom: test simulations **r03_i008**, **r03_i018**, **r03_i021**, and **r03_i023**, representing no break, almost break, almost break, and no break cases, respectively. Results for snapshot at time step (tlast-2) are shown on the left and time step tlast on the right.
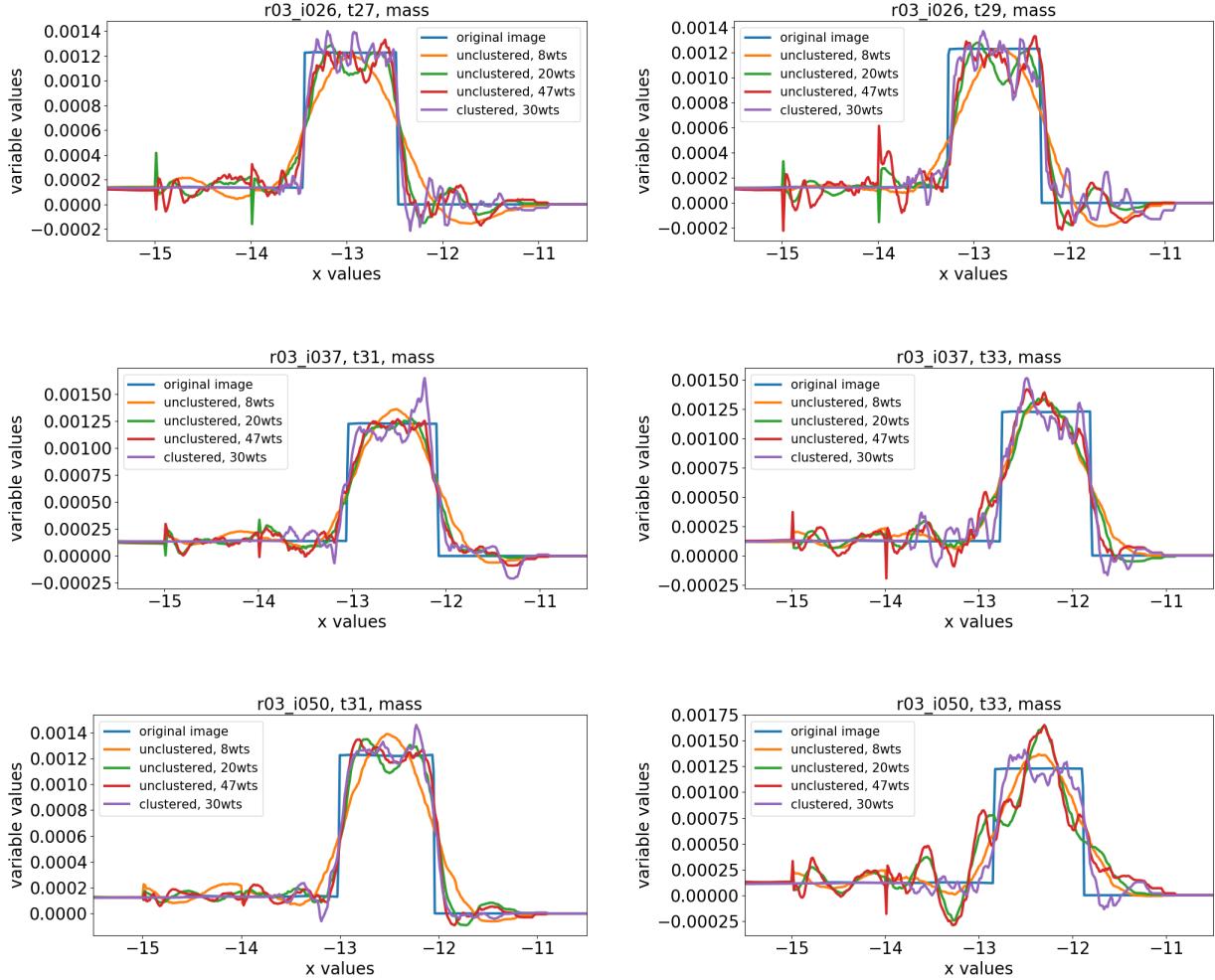
Figure 11: The y-lineouts for comparing the reconstruction of the *mass* variable both with and without clustering. From top to bottom: test simulations **r03_i026**, **r03_i037**, and **r03_i050**, all representing break cases. Results for snapshot at time step (tlast-2) are shown on the left and time step tlast on the right.

5.7 Discussion

We next summarize our observations on the reconstructed results presented for the mass and the x-momentum variables in Figures 10 and 11, and in Appendices C through F. These results indicate the following:

- For the mass variable, the reconstructions of the plate region using the linear surrogate are usually worse than the locally-linear surrogate. This is observed visually in the reconstructed snapshots and more clearly in the y-lineouts.

  When we use all snapshots in the linear surrogate, the location of the plate at $t = 0$ is captured prominently in the early eigen-snapshots. However, all the test snapshots are at late time, when the plate has moved to the right from its original location. Therefore, any errors in the predicted values of the early weights, when multiplied by the corresponding eigen-snapshots, appear as vertical streaks in the region to the left of the plate in the reconstructed snapshots. In contrast, for the locally-linear surrogate, the location of the plate in the eigen-snapshots is constrained to a smaller range of x values and closer to where we might expect the plate to be at late time. This is to be expected as the snapshot matrix in the locally-linear surrogate includes only the late time snapshots, instead of all the snapshots in the linear surrogate. The result is better quality reconstruction with the locally-linear surrogate.

22

The creation of the locally-linear surrogate requires the additional step of clustering of the snapshots. However, the facts that the calculation of the SVD is for a smaller snapshot matrix and the Gaussian process models are built for a smaller data set, make the creation of the locally-linear surrogate computationally faster than the linear surrogate.

- For both the mass and the x-momentum variables, the reconstruction at time step (tlast-2) is often better than at time step tlast.

  There are two contributing factors. First, the weights corresponding to higher indices at time step (tlast-2) are often smaller than the corresponding weights at time step tlast. So, when both snapshots are reconstructed with the same number of weights, ignoring the weights with higher indices, more useful information is ignored at time step tlast than at time step (tlast-2), resulting in better reconstruction of the latter. Second, the error in prediction of the weights at time step (tlast-2) is usually smaller than at time step tlast. This is to be expected. When we consider the sample points in the input region formed by the three simulation parameters and the time step, the point at the last time step is on the boundary of this region, while the point at (tlast-2) time step is near, but not at the boundary. Weight predictions at the boundary points are usually less accurate as they have fewer neighboring points around them.

- However, for the two no-break simulations, r03_i008 and r03_i023, the reconstructions of the mass and x-momentum variables, at both the tlast and the (tlast-2) time steps are poor.

  These two simulations are near the boundary of the region formed by the training data in the space of the three input parameters, as shown in Figure 7. Simulation r03_i023 also has the smallest jet tip velocity. With fewer neighbors around these points, the weight predictions from the Gaussian process models are less accurate, resulting in poor quality reconstructions.

- For the mass variable, we see different effects as we change the number of weights used in reconstruction with the linear surrogate. These effects are best understood through the y-lineplots. When the number of weights is small (=8), though the curve is very smooth, it is a poor fit to the plate profile, and extends far to the left of the plate. As the number of weights increases to 20 and then 47, the curves become a better fit to the profile of the plate and extend less to the left of the plate, but appear less smooth.

  There are several competing factors responsible for this behavior. The eigen-snapshots indicate that at a fixed y coordinate the values flip between positive and negative, with fewer sign changes in the initial eigen-snapshots and more in the later ones, somewhat akin to a Fourier series. This accounts for the smooth curve in the y-lineout at few weights, which becomes more jagged as the number of weights is increased and the "higher-frequency" eigen-snapshots are used to approximate the plate, which can be seen as a square wave. As we increase the number of weights, the approximation to the plate location becomes better, and the y-lineout curve, which extends far to the left of the plate at 8 weights, moves closer to the plate at 20 and 47 weights. However, the weights between 8 and 15 are often predicted with high error, which shows up as wiggles in the curve to the left of the plate. In contrast, for the locally-linear surrogate, the better localization of plate location in the eigen-snapshots and the lower error in weight prediction, lead to a better match of the reconstructed curve to the actual plate profile.

- For the mass variable, the reconstructed snapshots have negative values, which appears physically incorrect. This is due to the limited number of eigen-snapshots used in reconstruction. Using all eigen-snapshots would result in near perfect reconstruction and all positive values (to within floating point errors) for the mass variable.

- For the x-momentum variable, for which we only present results generated with the locally-linear surrogate, we observe certain differences with the mass variable. The values of both the variable and the weights are smaller for x-momentum. In most cases, the weights for x-momentum go rapidly to zero, though the initial weights have higher uncertainty. The reconstruction quality based on the y-lineplots indicates that the no-break cases and some of the break cases could be improved.

- We observe that in our problem, as the focus is on what happens to the plate and the jet at late time, we could have reduced the size of each snapshot further by focusing on just the region around the plate. However, such an option is problem dependent and may not be available in general.

- Finally, we consider to what extent we can address the eventual goals of this effort, namely, for a new point in the simulation input space, is it possible to use the surrogate to determine if the plate breaks at late time, to identify the final location of the plate, and to obtain the speed of the jet as it exits the plate in cases where the plate breaks.

  We address these questions using the reconstructed region around the plate and the y-lineouts for the cases where the snapshots have good reconstruction quality. Understandably, the reconstructed snapshots are an approximation to the actual snapshots, especially when a small number of weights are used. However, the y-lineouts indicate that it should be possible to obtain a good estimate of the plate location at late time by applying gradient-based image segmentation techniques. To determine whether the plate is a break, almost-break, or no-break case, we can look at the bottom region of the plate to see if it has detached from the bottom and if we can see the jet on the other side. We observe that it may be harder to distinguish between the no-break and the almost-break cases, but this may be due to the poor reconstruction of the no-break cases. However, the break cases appear to be easy to identify. This is despite the fact that the boundary of the plate is not as sharply defined in the reconstruction as we have ignored the weights at higher indices.

  Understanding whether we can meet our goals for the no-break cases, which are poorly reconstructed given our training data, we will first need to improve these cases using the ideas discussed next.

Overall, we observe that the quality of the reconstructed test snapshots is a combination of several factors, including the suitability of the initial eigen-snapshots at capturing the plate location at late time, the error in the prediction of the weights, and the number of weights used in the reconstruction. This gives us some suggestions for generating better quality reconstructions, without increasing the number of simulations:

- For our problem, the locally-linear surrogate created using only the late time snapshots, gives better results than the linear surrogate created using all snapshots from the 45 simulations. We expect this result to hold in general. By clustering the snapshots, and building linear surrogates for each cluster, we generate a better basis for the snapshots in that cluster, leading to more accurate predictions for new snapshots in that cluster. This requires the identification of a cluster for the new snapshot, which can be obtained from the cluster assignment of all the snapshots. It also suggests that for our problem, other non-linear transform methods may be worth exploring.

- To reduce the error in the prediction of the weights, we want the test point to be in the interior of the region formed by the training data in the space of the three input parameters and time step.

  Therefore, if our interest is in predicting what happens at late time steps, as in our jet-HE interaction problem, we should run the simulations used to create the training data for a few more time steps, so the test points lie in the interior of the region that forms the input to the Gaussian process models.

  This also means that the test points should lie in the interior of the space of the simulation input parameters. Since the locations of the test points may not always be known before the training data are generated, one approach to ensure that the test points are not too far from the training data is to cover the region of interest with random points far from each other. We accomplish this using the best candidate sampling. In addition, decisions not to add points in some regions of the input space, should be taken with care; our decision to exclude points in the high HE-length, low jet tip velocity region led to poor prediction for the no-break test cases. Further, it may be desirable to set aside a number of simulations to be run once the locations of the test points are known.

- It still remains a challenge to determine the number of weights to use for reconstruction, especially as the quality of the weight prediction could be poor when the number of simulations is small. Using too few weights would result in poor reconstruction of sharp changes in the data, such as the plate boundary, while using more weights might introduce errors when the weight predictions are not accurate. Ideally, we want the weight values to decrease rapidly as it would indicate that a small number of initial weights is necessary for reconstruction. However, this may not be the case when there is large variation in the data that is not captured sufficiently by a small number of simulations. It therefore appears that we may require experimentation with different number of weights, with the number selected possibly varying with each test snapshot.

6   Conclusions

In this report, we considered the problem of jet-HE interaction to determine if it is possible to build accurate, spatio-temporal surrogates when we can run only a small number of simulations to create a training data set. We showed how to process a data set where the size of the computational domain varies with each simulation and each snapshot has over two million grid points. Our results showed that a locally-linear surrogate, which builds separate surrogate models using groups of similar snapshots, is more accurate than one which builds a single surrogate using all the snapshots. We also identified other simple ways to improve the quality of surrogates when we are constrained to run only a limited number of simulations. These include better sampling of the training data points in the simulation input parameter space to cover the region uniformly so no test point is too far from a training point; selecting, if possible, the locations of the training data points such that the test snapshots are not near the boundary of the region defined by the training data; and setting aside a part of the simulation budget to run a few additional simulations once the test data points have been identified.

7   Acknowledgment

References

[1] N. Alexandris, S. Gupta, and N. Koutsias. Remote sensing of burned areas via PCA, Part 1; centering, scaling and EVD vs SVD. *Open Geospatial Data, Software and Standards*, 2, 2017. doi: https://doi.org/10.1186/s40965-017-0028-1.

[2] D. Amsallem, M. J. Zahr, and C. Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10):891–916, 2012. doi: https://doi.org/10.1002/nme.4371.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8.

[4] G. Aversano, A. Bellemans, Z. Li, A. Coussement, O. Gicquel, and A. Parente. Application of reduced-order models based on PCA and Kriging for the development of digital twins of reacting flow applications. *Computers & Chemical Engineering*, 121:422–441, 2019. doi: https://doi.org/10.1016/j.compchemeng.2018.09.022.

[5] A. Bērzinš, J. Helmig, F. Key, and S. Elgeti. Standardized non-intrusive reduced order modeling using different regression models with application to complex flow problems, 2021. arXiv:2006.13706v2 [physics.comp-ph].

[6] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[7] J. Cadima and I. T. Jolliffe. On relationships between uncentred and column-centred principal component analysis. *Pak J. Statist.*, 25:473–503, 2009.

[8] P. Constantine and D. Gleich. Tall and skinny QR factorizations in MapReduce architectures. In *MapReduce'11 - Proceedings of the 2nd International Workshop on MapReduce and Its Applications*, January 2011. doi: https://doi.org/10.1145/1996092.1996103.

[9] S. Dasgupta and A. Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2002. doi: https://doi.org/10.1002/rsa.10073.

[10] K. Fukami, T. Nakamura, and K. Fukagata. Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data. *Physics of Fluids*, 32:095110, 2020.

[11] D. Higdon, J. Gattiker, B. Williams, and M. Rightley. Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, 103(482):570–583, 2008. doi: https://doi.org/10.1198/016214507000000888.

[12] G.E. Hinton, P. Dayan, and M. Revow. Modeling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74, 1997. doi: https://doi.org/10.1109/72.554192.

[13] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[14] I. T. Jolliffe and J. Cadima. Principal component analysis: A review and recent developments. *Phil. Trans. R. Soc. A.*, 374, 2016. doi: http://doi.org/10.1098/rsta.2015.0202.

[15] C. Kamath. Intelligent sampling for surrogate modeling, hyperparameter optimization, and data analysis. *Machine Learning with Applications*, 9:100373, 2022. doi: https://doi.org/10.1016/j.mlwa.2022.100373.

[16] C. Kamath and Y. J. Fan. Regression with small data sets: A case study using code surrogates in additive manufacturing. *Knowl. Inf. Syst.*, 57(2):475–493, November 2018. doi: https://doi.org/10.1007/s10115-018-1174-1.

[17] C. Kamath and J. S. Franzman. Spatio-temporal surrogates for interaction of a jet with high explosives: Part II - Clustering extremely high-dimensional grid-based data. Technical Report LLNL-TR-850159, Lawrence Livermore National Laboratory CA., June 2023.

[18] N. Kambhatla and T. K. Leen. Fast non-linear dimension reduction. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993. URL https://proceedings.neurips.cc/paper/1993/file/d554f7bb7be44a7267068a7df88ddd20-Paper.pdf.

[19] N. Kambhatla and T. K. Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9:1493–1516, 1997.

[20] N. Kambhatla and T.K. Leen. Fast nonlinear dimension reduction. In *IEEE International Conference on Neural Networks*, pages 1213–1218 vol.3, 1993. doi: https://doi.org/10.1109/ICNN.1993.298730.

[21] M. Kirby and L. Sirovich. Application of the Karhunen-Loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990. doi: https://doi.org/10.1109/34.41390.

[22] M. Kirby, J. Boris, and L. Sirovich. An eigenfunction analysis of axisymmetric jet flow. *Journal of Computational Physics*, 90:98–122, 1990.

[23] J. A. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction.* Springer, New York, NY, USA, 2007.

[24] P. Li, T. J. Hastie, and K. W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 287–296, New York, NY, USA, 2006. Association for Computing Machinery. doi: https://doi.org/10.1145/1150402.1150436.

[25] H. V. Ly and H. T. Tran. Modeling and control of physical processes using proper orthogonal decomposition. *Mathematical and Computer Modelling*, 33(1):223–236, 2001. doi: https://doi.org/10.1016/S0895-7177(00)00240-5. Computation and control VI proceedings of the sixth Bozeman conference.

[26] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[27] D. P. Mitchell. Spectrally optimal sampling for distribution ray tracing. *Computer Graphics*, 25(4):157–164, 1991.

[28] M. Mitry. Reduced Order Surrogate Modelling (ROSM) of high dimensional deterministic simulations. Master's thesis, Department of Aerospace Engineering, University of Toronto, 2014. Available from `http://hdl.handle.net/1807/68047`.

[29] D. Rajaram, C. Perron, T. G. Puranik, and D. N. Mavris. Randomized algorithms for non-intrusive parametric reduced order modeling. *AIAA Journal*, 58(12):5389–5407, 2020. doi: https://doi.org/10.2514/1.J059616.

[30] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning.* MIT Press, Cambridge, MA, 2006.

[31] L. Sirovich. Turbulence and the dynamics of coherent structures part i: coherent structures. *Quarterly of Applied Mathematics*, 45:561–571, 1987.

[32] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *J Opt Soc Am A.*, 4:519–524, 1987. doi: https://doi.org/10.1364/josaa.4.000519.

[33] R. Swischuk, L. Mainini, B. Peherstorfer, and K. Willcox. Projection-based model reduction: Formulations for physics-based machine learning. *Computers & Fluids*, 179:704–717, 2019. doi: https://doi.org/10.1016/j.compfluid.2018.07.021.

[34] The HDF Group. Hierarchical Data Format, version 5, 1997-2023. https://www.hdfgroup.org/HDF5/.

[35] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 01 1991. doi: https://doi.org/10.1162/jocn.1991.3.1.71.

[36] M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991. doi: https://doi.org/10.1109/CVPR.1991.139758.

[37] R. A. van de Geijn and M. Myers. *Advanced Linear Algebra: Foundations to Frontiers.* Online book available at `www.ulaff.net`, 2023.

[38] P. Wu, J. Sun, X. Chang, W. Zhang, R. Arcucci, Y. Guo, and C. C. Pain. Data-driven reduced order model with temporal convolutional neural network. *Computer Methods in Applied Mechanics and Engineering*, 360:112766, 2020. doi: https://doi.org/10.1016/j.cma.2019.112766.

[39] W.W. Xing, V. Triantafyllidis, A.A. Shah, P.B. Nair, and N. Zabaras. Manifold learning for the emulation of spatial fields from computational models. *Journal of Computational Physics*, 326:666–690, 2016. doi: https://doi.org/10.1016/j.jcp.2016.07.040.

[40] S.-T. Yeh, X. Wang, C.-L. Sung, S. Mak, Y.-H. Chang, L. Zhang, C. F. Wu, and V. Yang. Common proper orthogonal decomposition-based spatiotemporal emulator for design exploration. *AIAA Journal*, 56:2429–2442, 2018. doi: https://doi.org/10.1016/j.jcp.2016.07.040.

[41] J. Yu, C. Yan, Z. Jiang, W. Yuan, and S. Chen. Adaptive non-intrusive reduced order modeling for compressible flows. *Journal of Computational Physics*, 397:108855, 2019. doi: https://doi.org/10.1016/j.jcp.2019.07.053.

A   Appendix: Data for x-momentum variable
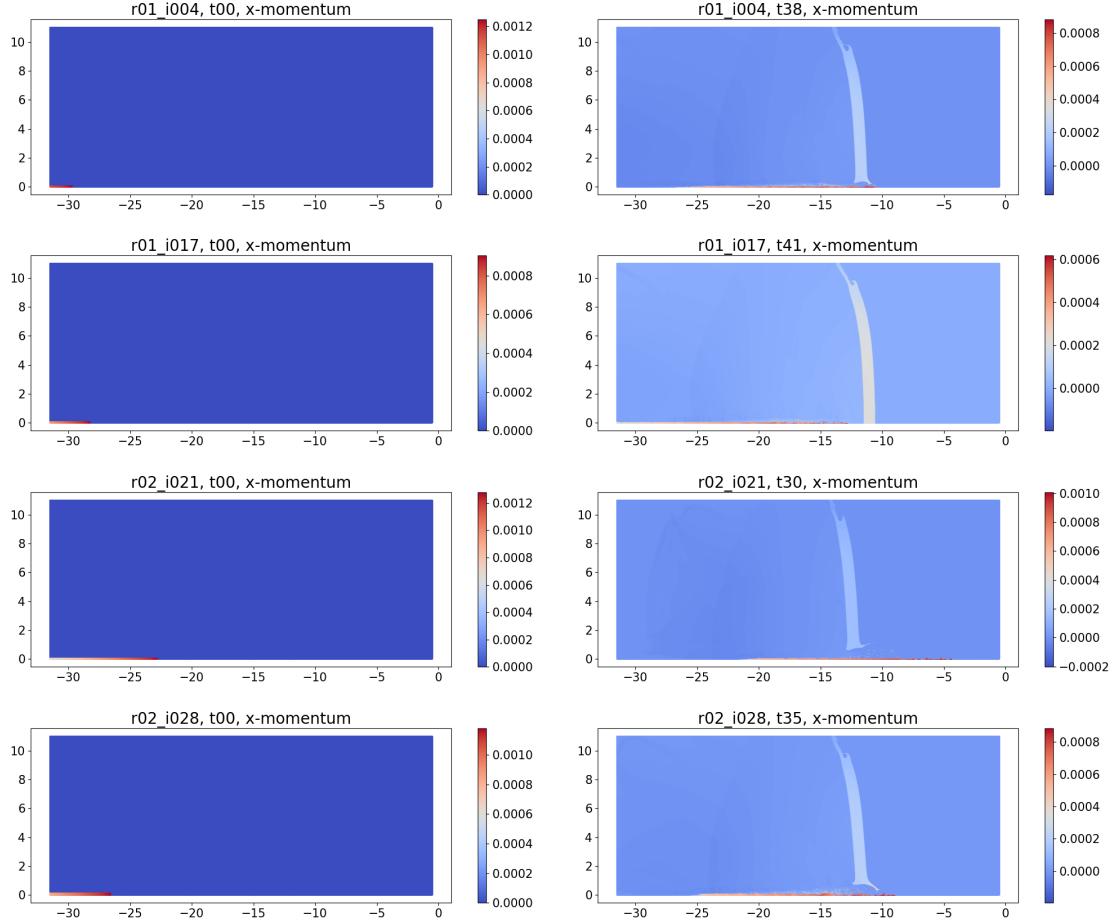
Figure 12: The variable *x-momentum*, after the snapshots have been aligned, cropped, and remapped, at the first time step (left column) and last time step (right column) for the four example simulations in Table 1. From top to bottom, simulations with keys r01_i004, r01_i017, r02_i021, r02_i028, respectively. The first time step has value zero for most of the domain, except the jet along the bottom on the left side of the region. The color bars are different between simulations and across time steps.
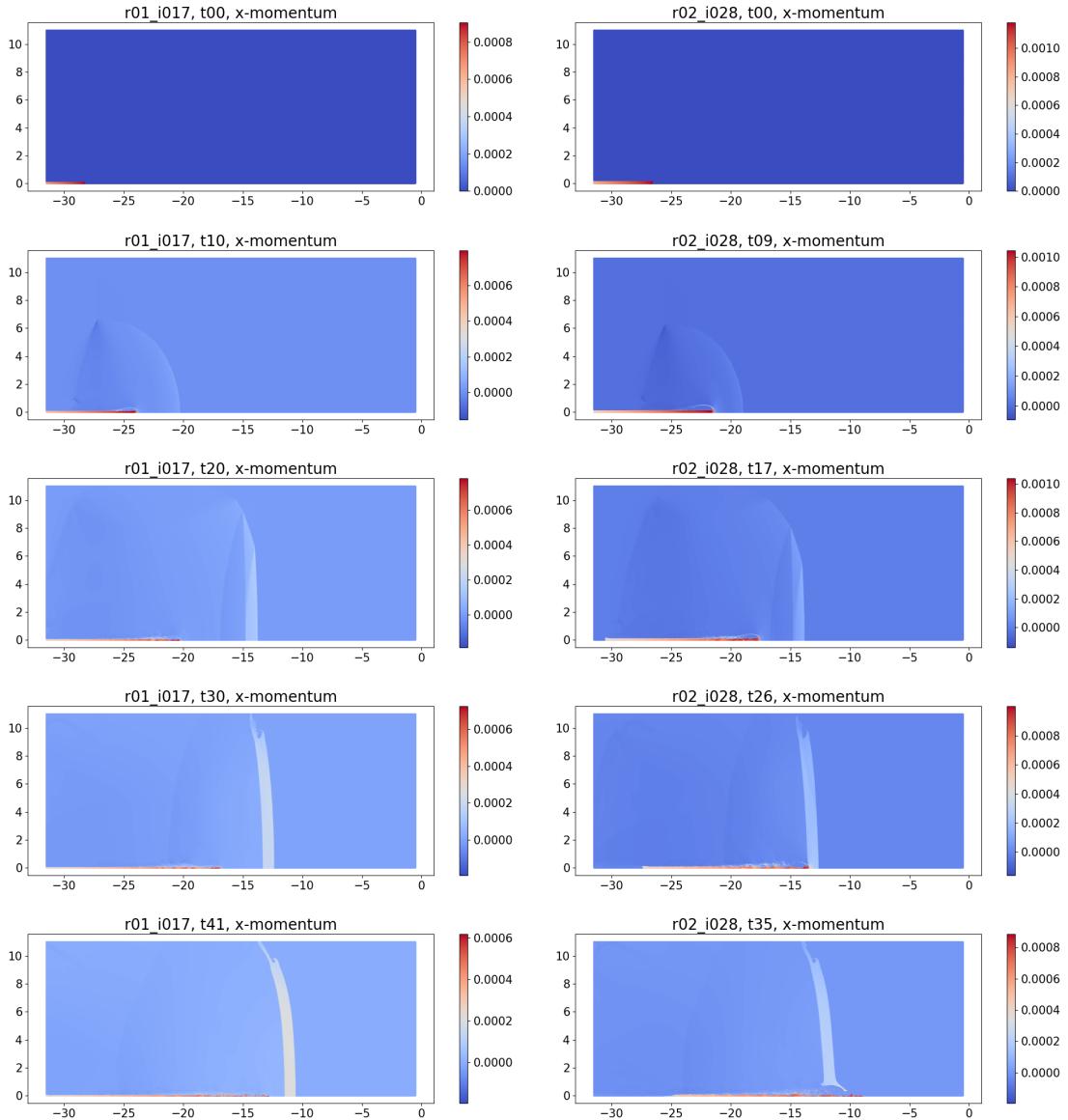
Figure 13: The variable *x-momentum*, after the snapshots have been aligned, cropped, and remapped, at different time steps in two simulations showing the evolution of the data over time. Left: key r01_i017 (no break case) at time steps t0, t10, t20, t30, and t41. Right: key r02_i028 (break case) at time steps t0, t09, t17, t26, and t35. The color bars are different between simulations and across time steps.

B   Appendix: Data for y-momentum variable
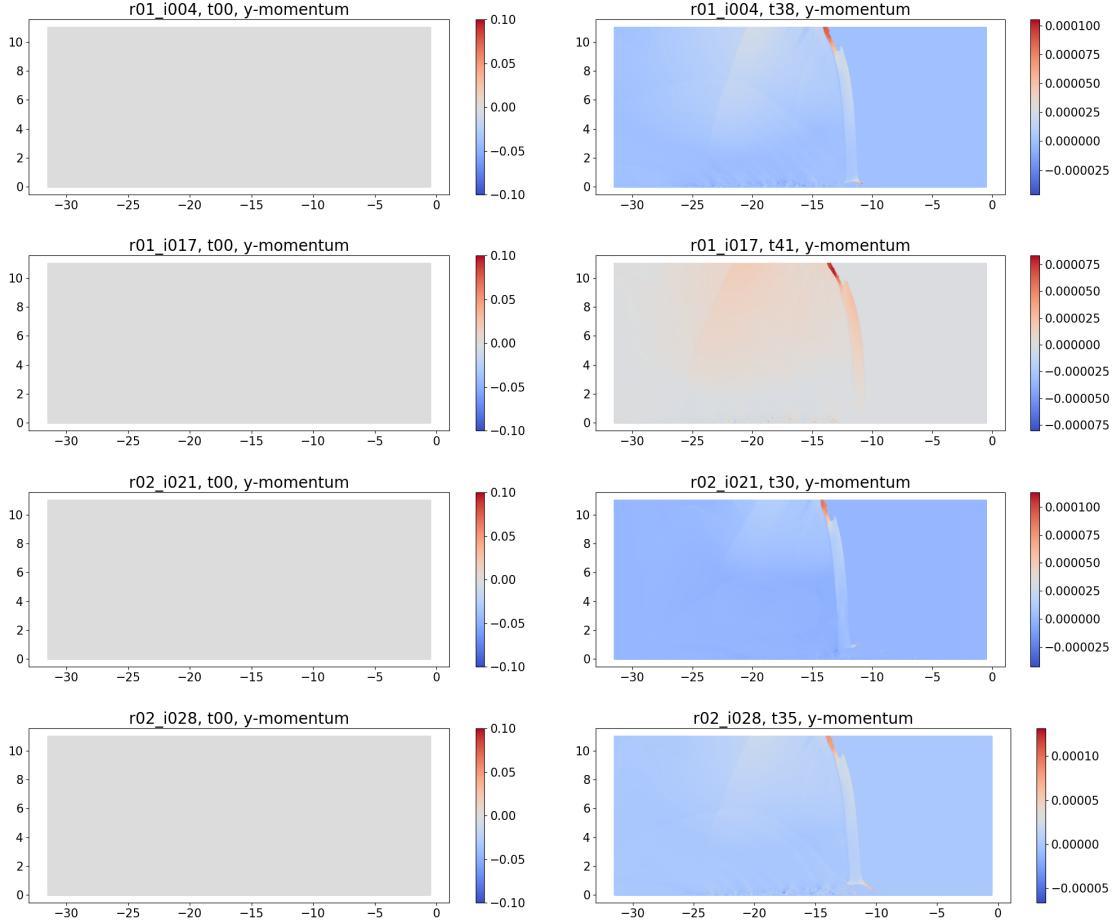
Figure 14: The variable *y-momentum*, after the snapshots have been aligned, cropped, and remapped, at the first (left column) and last (right column) time steps for the four example simulations in Table 1. From top to bottom, simulations with keys r01_i004, r01_i017, r02_i021, r02_i028, respectively. The first time step has value zero for most of the domain. Unlike the x-momemtum in Figure 12, there is no y-momentum for the jet at initial time. The color bars are different between simulations and across time steps.
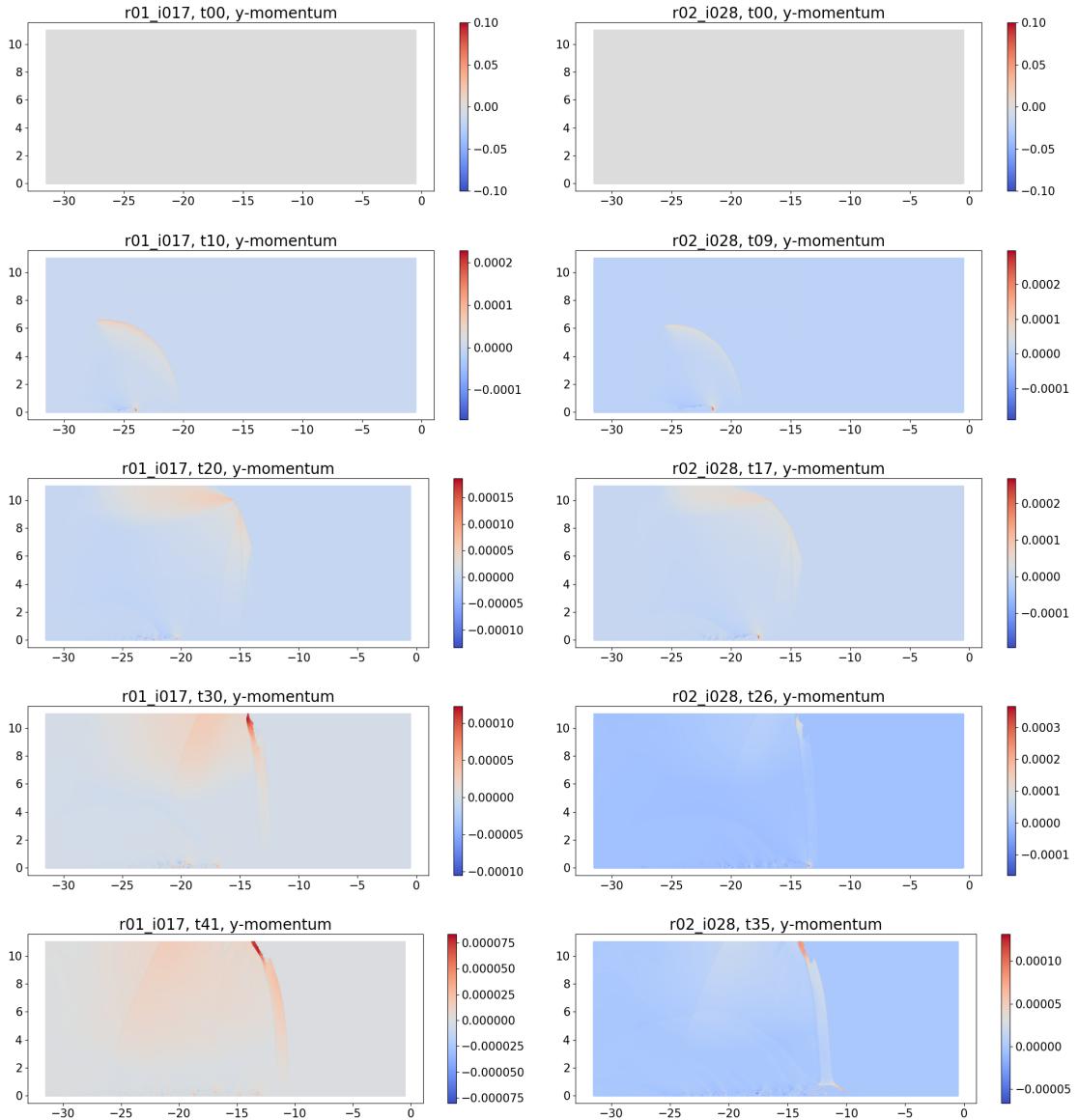
Figure 15: The variable *y-momentum*, after the snapshots have been aligned, cropped, and remapped, at different time steps in two simulations showing the evolution of the data over time. Left: key r01_i017 (no break case) at time steps t0, t10, t20, t30, and t41. Right: key r02_i028 (break case) at time steps t0, t09, t17, t26, and t35. The color bars are different between simulations and across time steps.

C Appendix: Eigen-snapshots for mass variable, with and without clustering
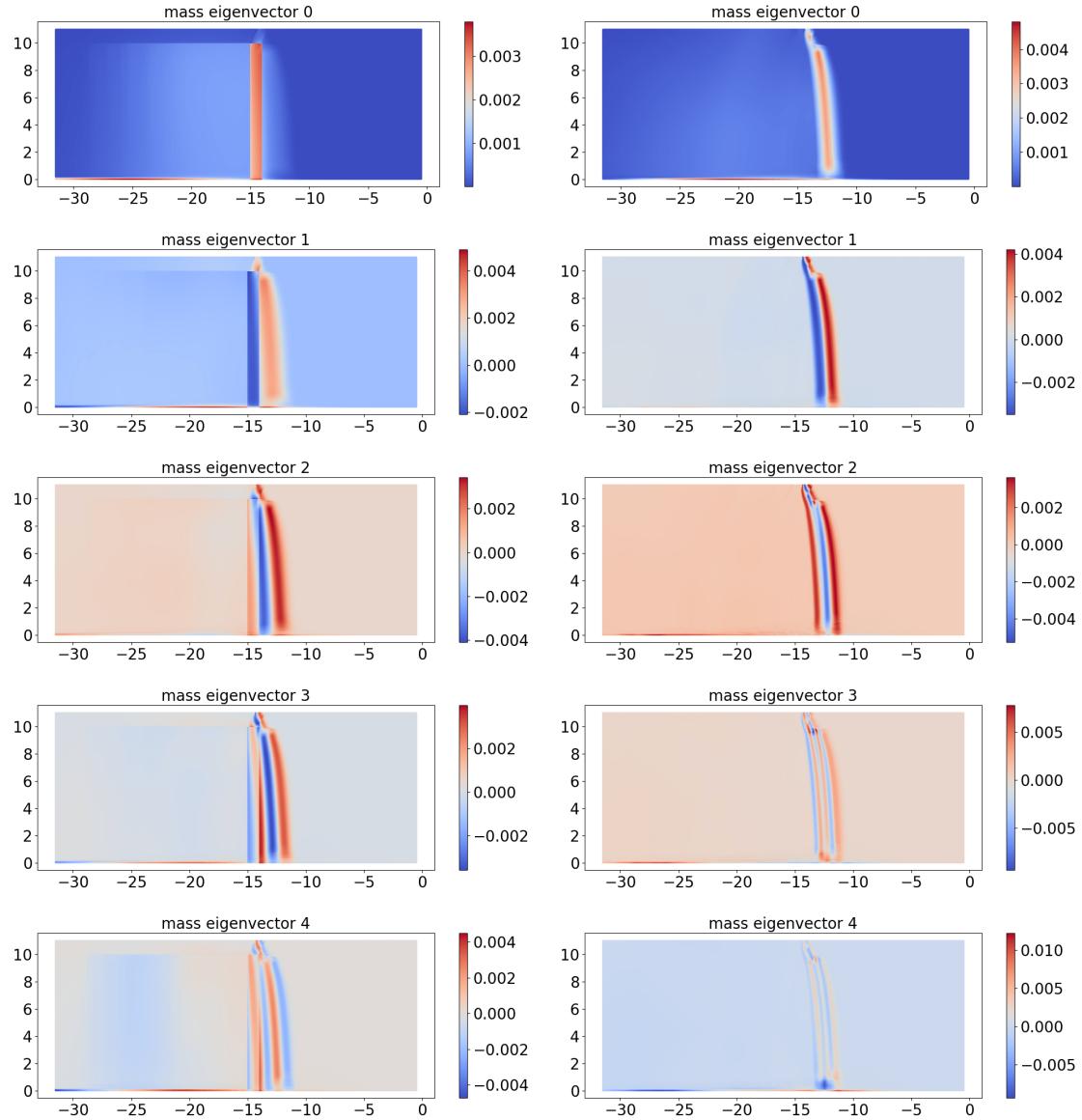
Figure 16: Eigen-snapshots (0-4) for variable *mass*, without clustering (on left) and with clustering, cluster 2, on right. The color bars are different across eigen-snapshots.
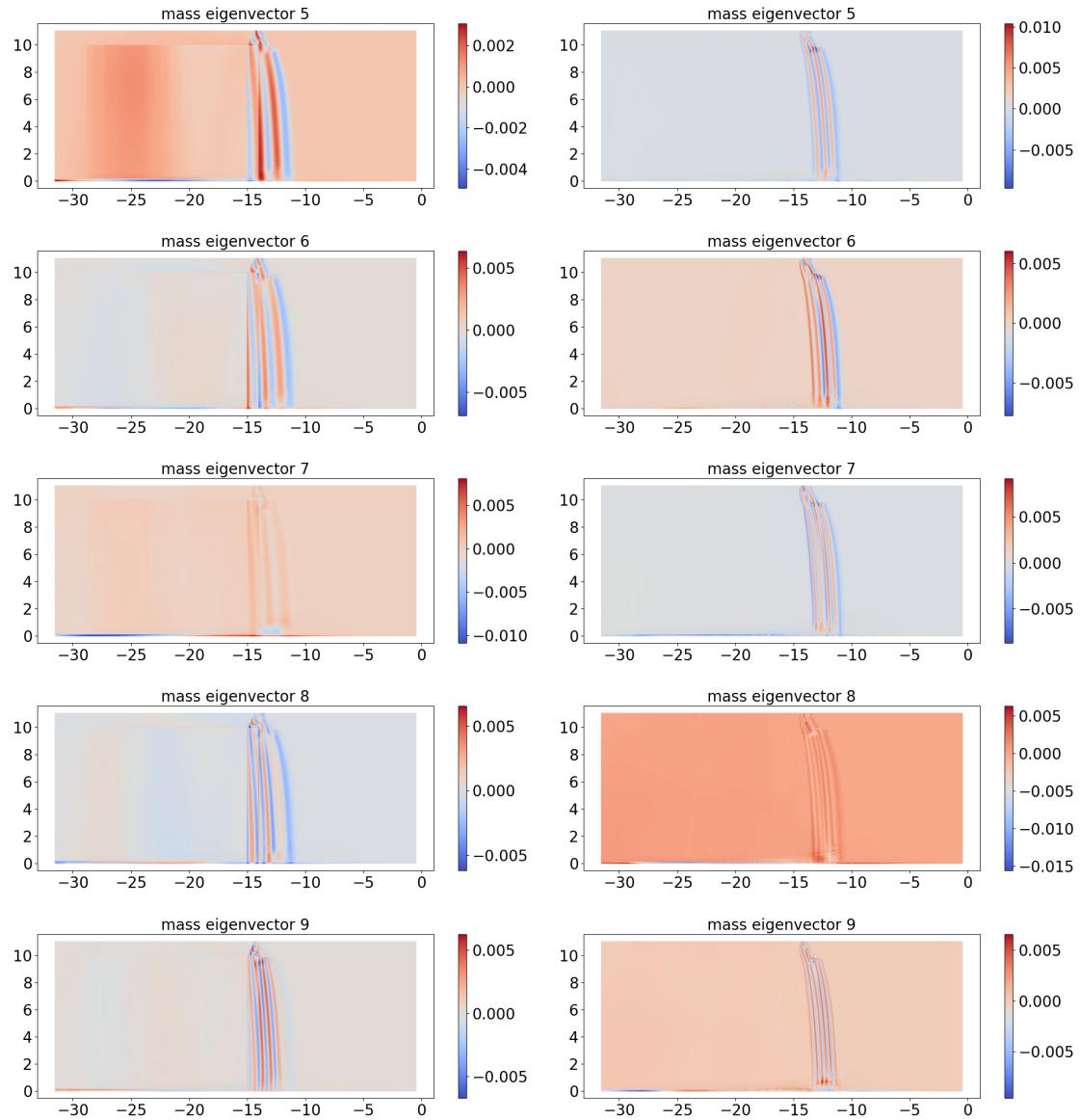
Figure 17: Eigen-snapshots (5-9) for variable *mass*, without clustering (on left) and with clustering, cluster 2, on right. The color bars are different across eigen-snapshots.

D Appendix: Reconstructed mass (before and after clustering) for all seven test cases
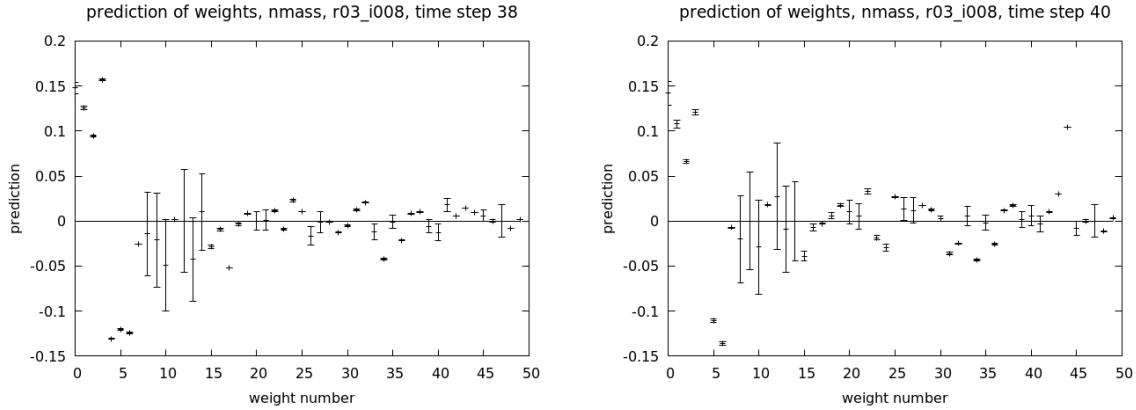
Figure 18: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i008** at time t38 (left) and t40 (right). Plots show the weight predictions (with 1 standard deviation error bar) for the first 50 weights.
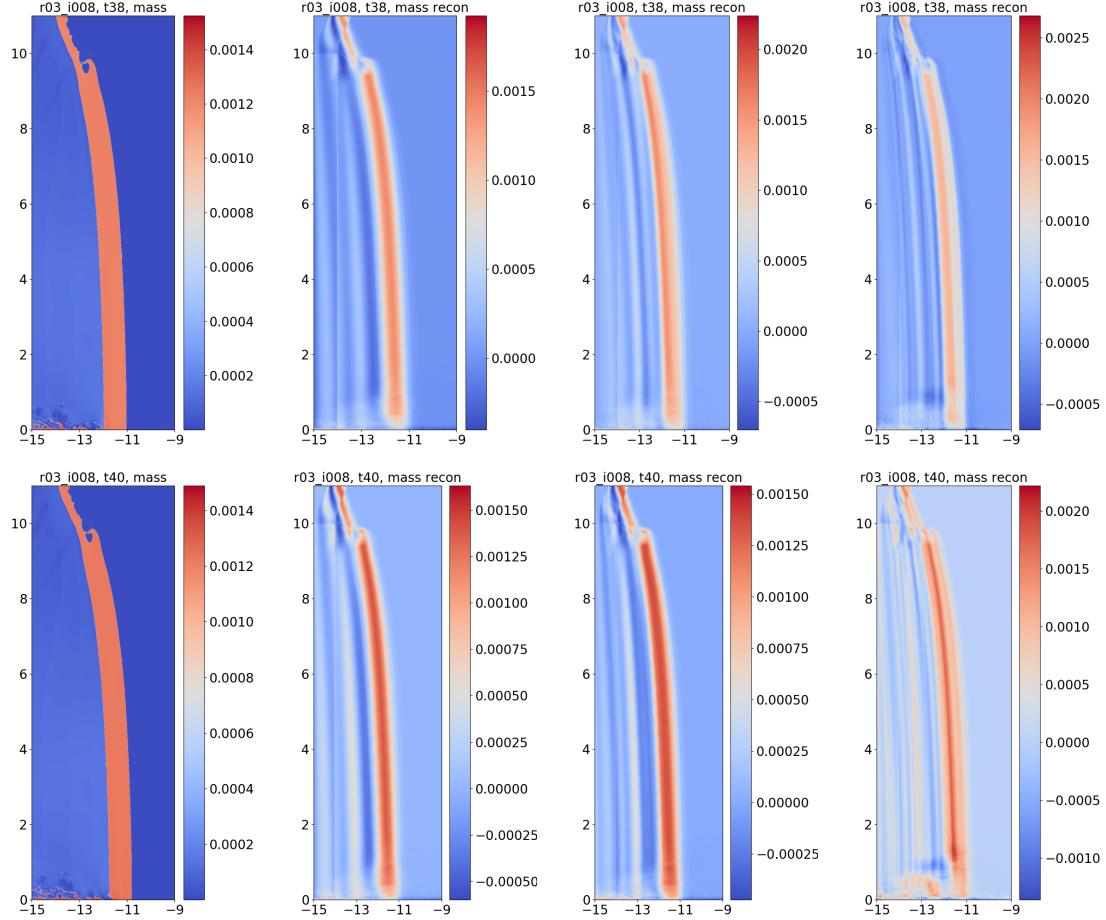

Figure 19: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i008** at time t38 (top) and t40 (bottom). The plate view shows from left to right: the original snapshot followed by the reconstruction using 8, 20, and 47 weights.
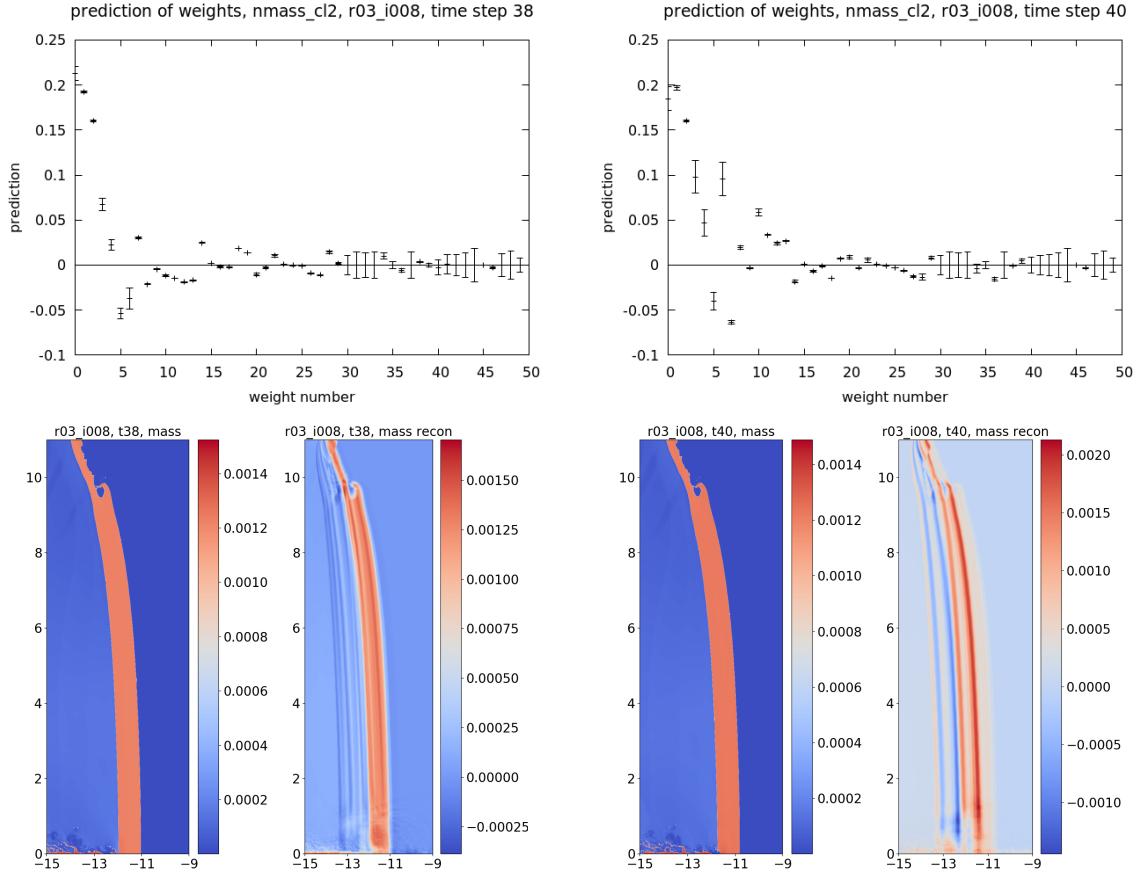
Figure 20: Reconstruction of the variable *mass*, after clustering, for test simulation **r03_i008** at time t38 (left) and t40 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 30 weights.
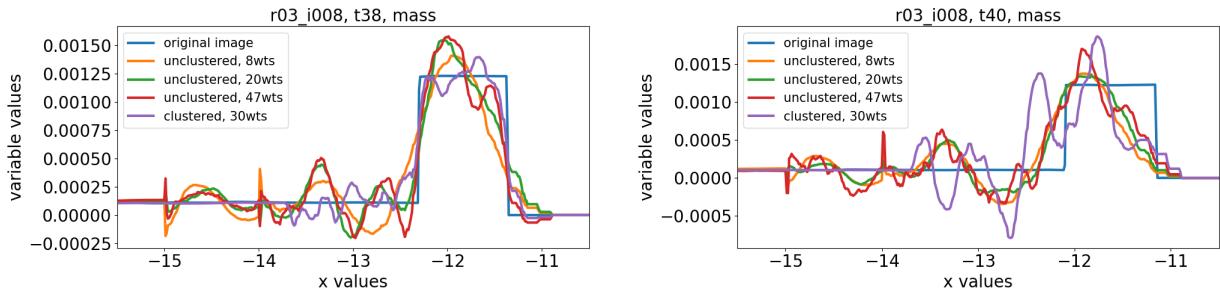

Figure 21: Comparison of the reconstruction of the variable *mass*, both with and without clustering, for test simulation **r03_i008** at time t38 (left) and t40 (right). The plots compare the value of the variable at $y = 6.0063$ for the different reconstructions to indicate how well they detect the plate boundary.
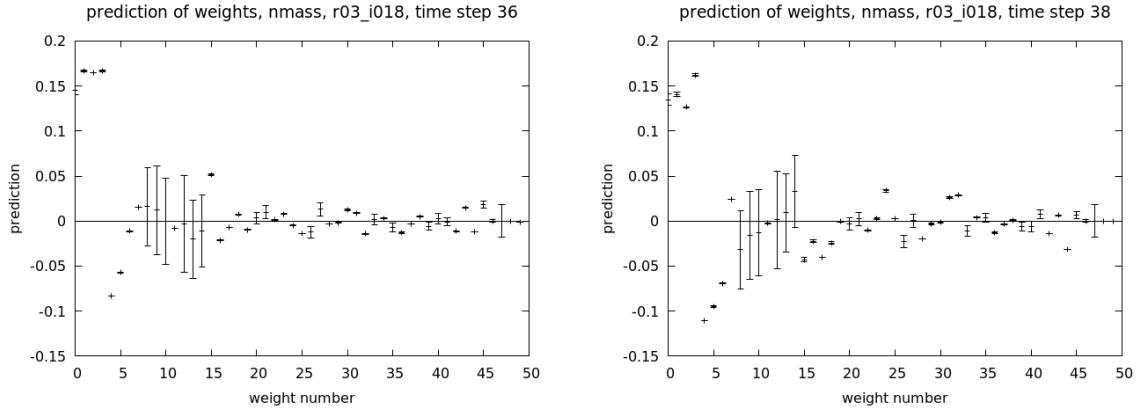
Figure 22: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i018** at time t36 (left) and t38 (right). Plots show the weight predictions (with 1 standard deviation error bar) for the first 50 weights.
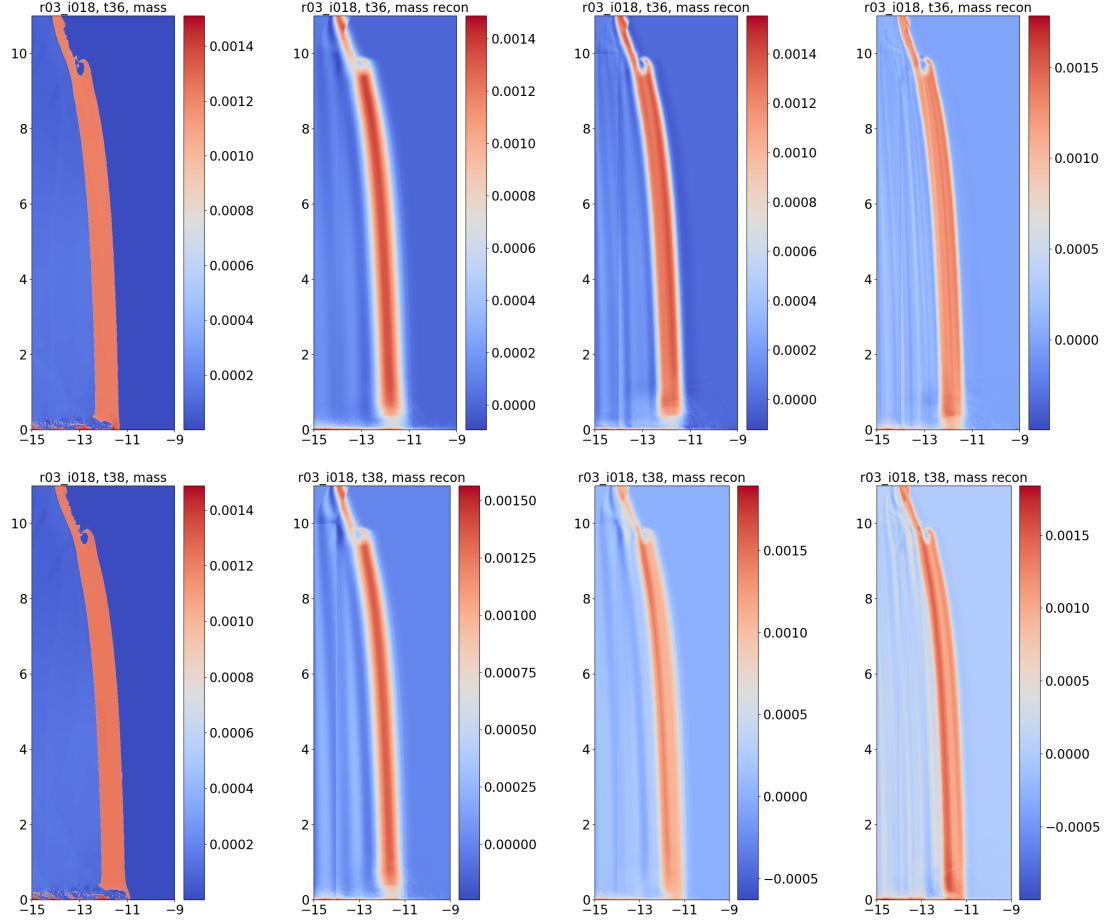
Figure 23: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i018** at time t36 (top) and t38 (bottom). The plate view shows from left to right: the original snapshot followed by the reconstruction using 8, 20, and 47 weights.
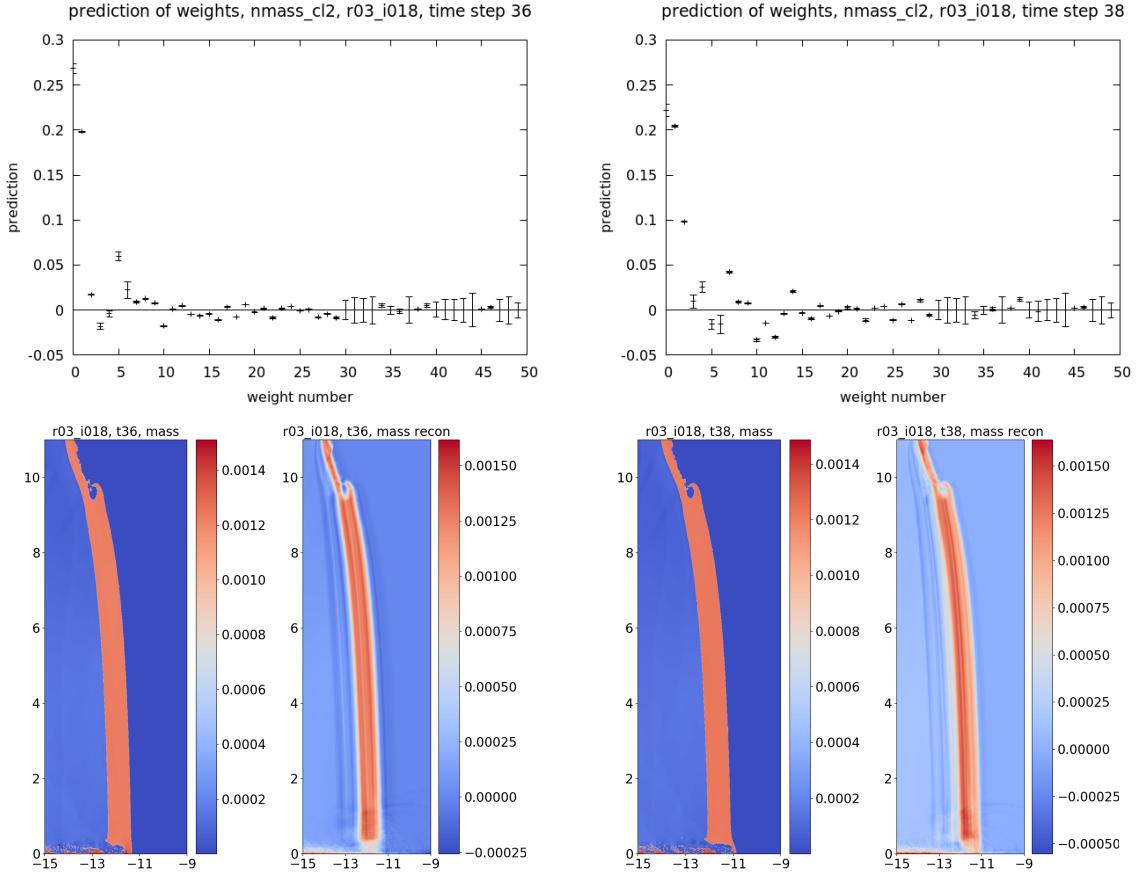
Figure 24: Reconstruction of the variable *mass*, after clustering, for test simulation **r03_i018** at time t36 (left) and t38 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 30 weights.
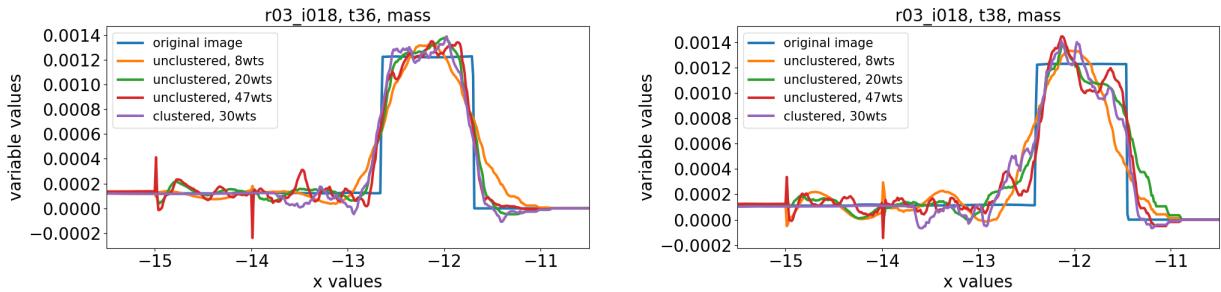
Figure 25: Comparison of the reconstruction of the variable *mass*, both with and without clustering, for test simulation **r03_i018** at time t36 (left) and t38 (right). The plots compare the value of the variable at $y = 6.0063$ for the different reconstructions to indicate how well they detect the plate boundary.
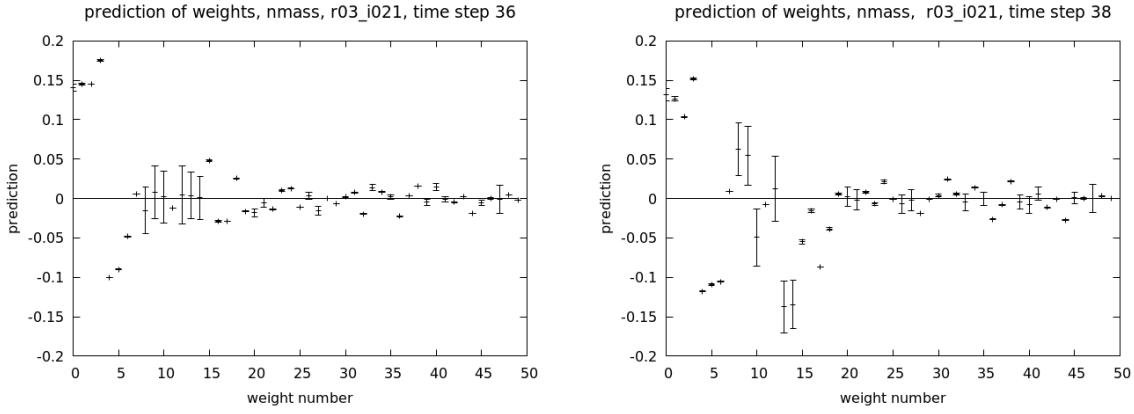
Figure 26: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i021** at time t36 (left) and t38 (right). Plots show the weight predictions (with 1 standard deviation error bar) for the first 50 weights.
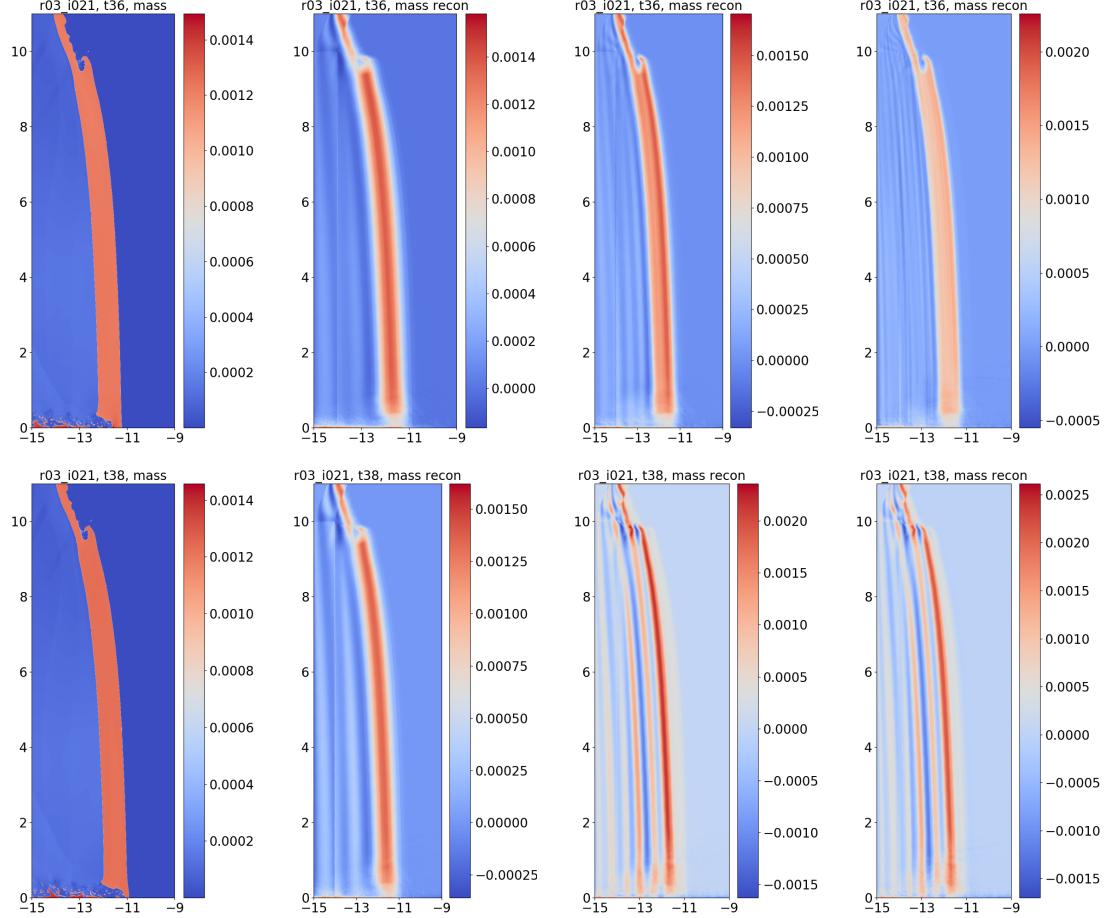

Figure 27: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i021** at time t36 (top) and t38 (bottom). The plate view shows from left to right: the original snapshot, followed by the reconstruction using 8, 20, and 47 weights.
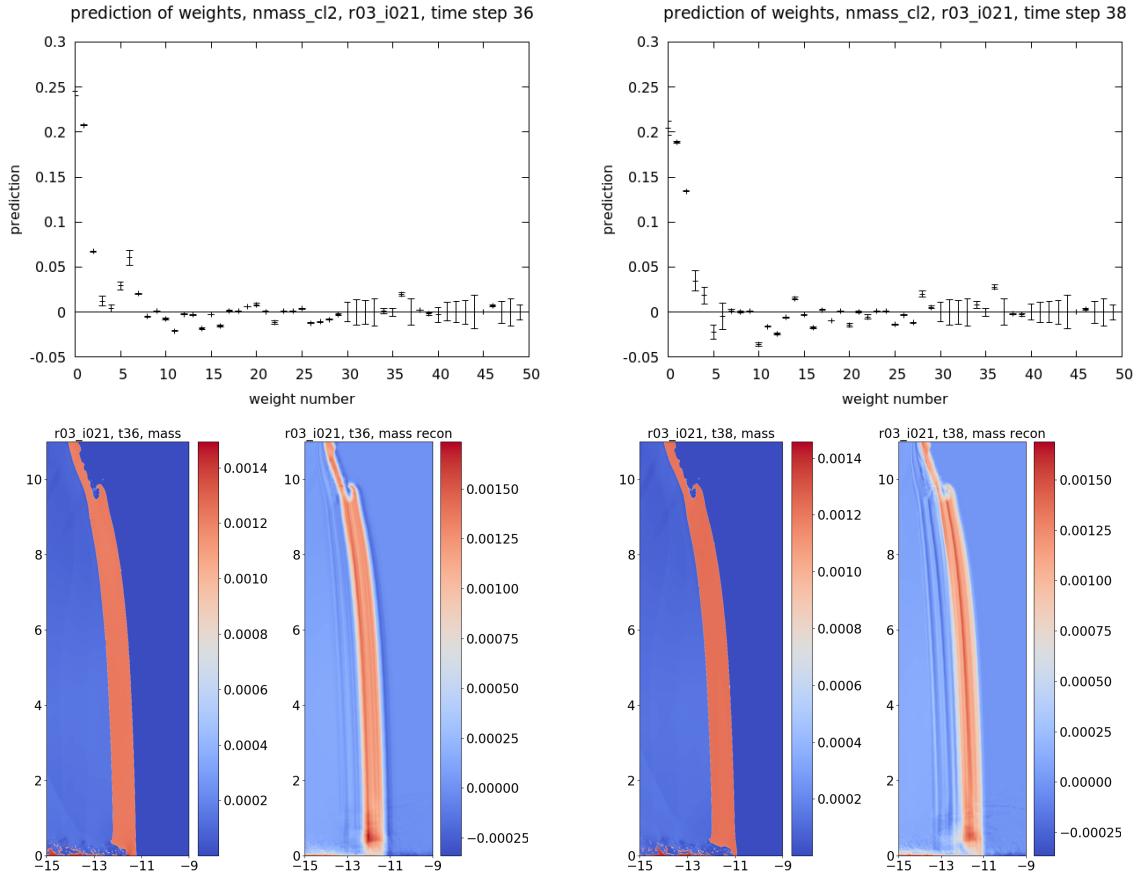
Figure 28: Reconstruction of the variable *mass*, after clustering, for test simulation **r03_i021** at time t36 (left) and t38 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 30 weights.
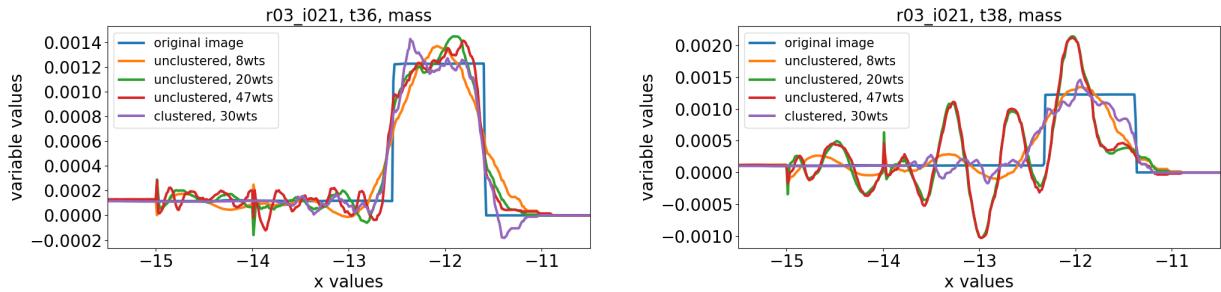

Figure 29: Comparison of the reconstruction of the variable *mass*, both with and without clustering, for test simulation **r03_i021** at time t36 (left) and t38 (right). The plots compare the value of the variable at $y = 6.0063$ for the different reconstructions to indicate how well they detect the plate boundary.
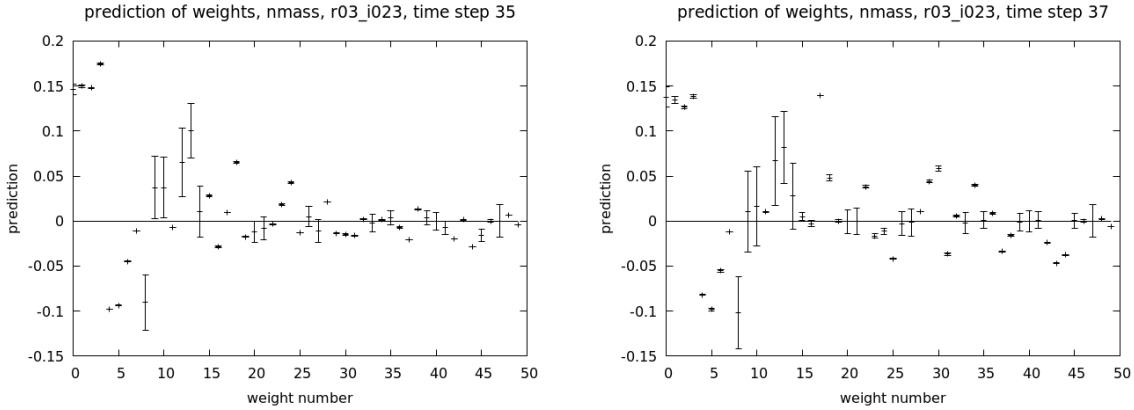
Figure 30: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i023** at time t35 (left) and t37 (right). Plots show the weight predictions (with 1 standard deviation error bar) for the first 50 weights.
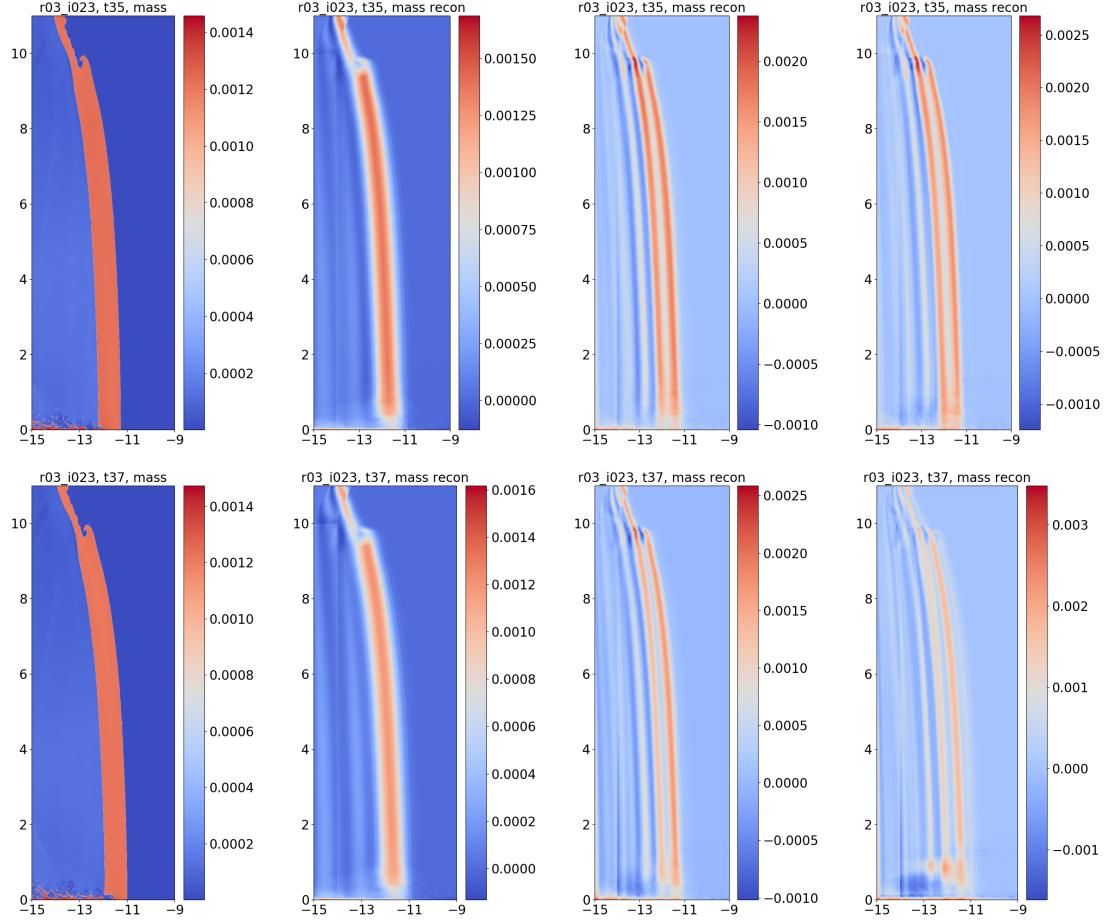

Figure 31: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i023** at time t35 (top) and t37 (bottom). The plate view shows from left to right: the original snapshot; followed by the reconstruction using 8, 20, and 47 weights.
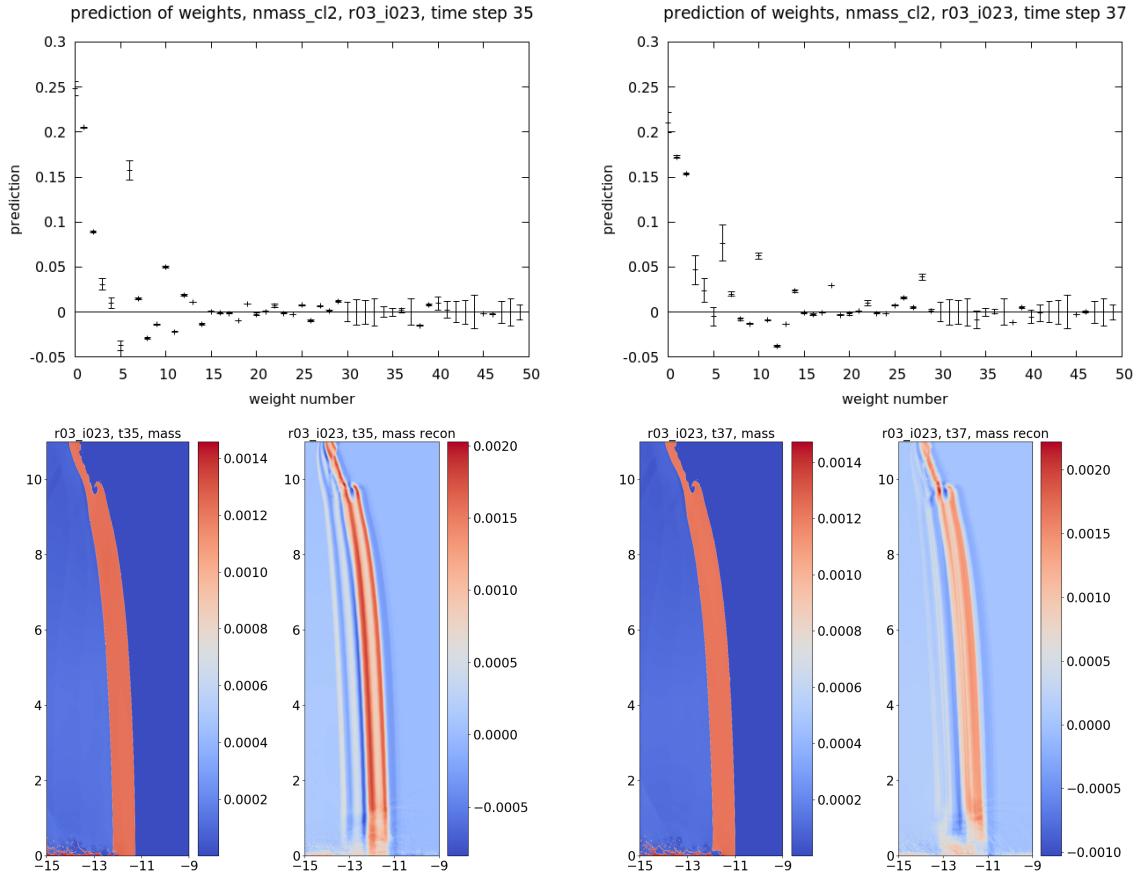
Figure 32: Reconstruction of the variable *mass*, after clustering, for test simulation **r03_i023** at time t35 (left) and t37 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 30 weights.
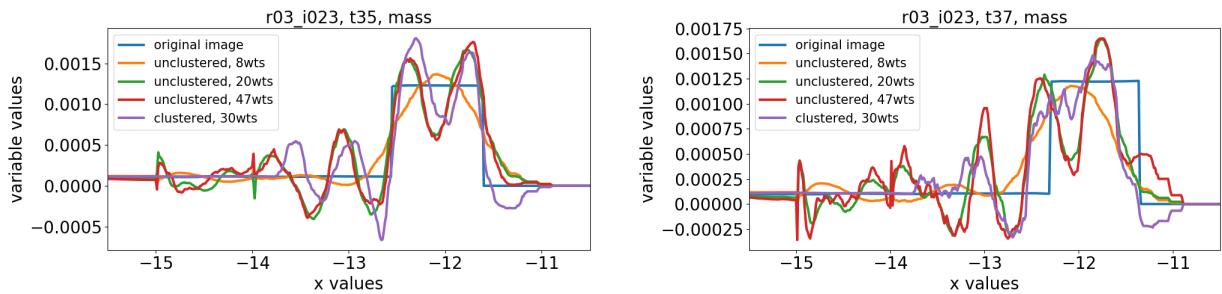

Figure 33: Comparison of the reconstruction of the variable *mass*, both with and without clustering, for test simulation **r03_i023** at time t35 (left) and t37 (right). The plots compare the value of the variable at $y = 6.0063$ for the different reconstructions to indicate how well they detect the plate boundary.
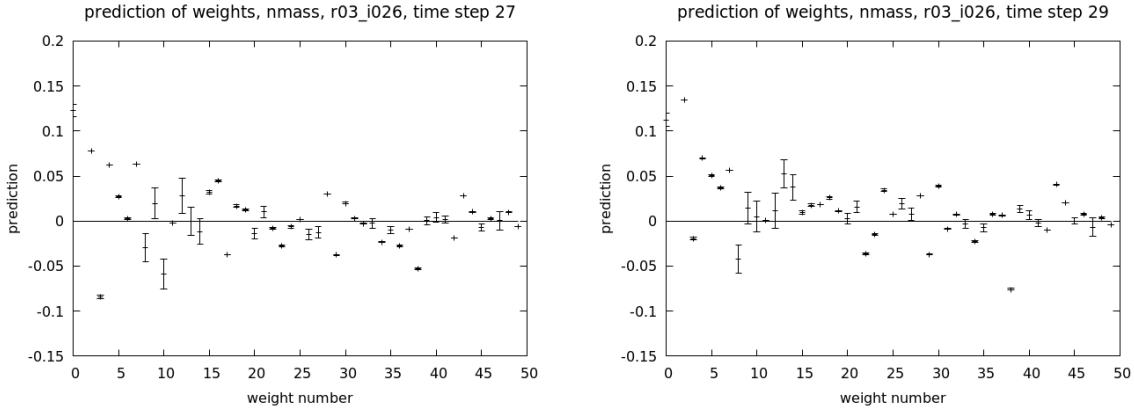
Figure 34: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i026** at time t27 (left) and t29 (right). Plots show the weight predictions (with 1 standard deviation error bar) for the first 50 weights.
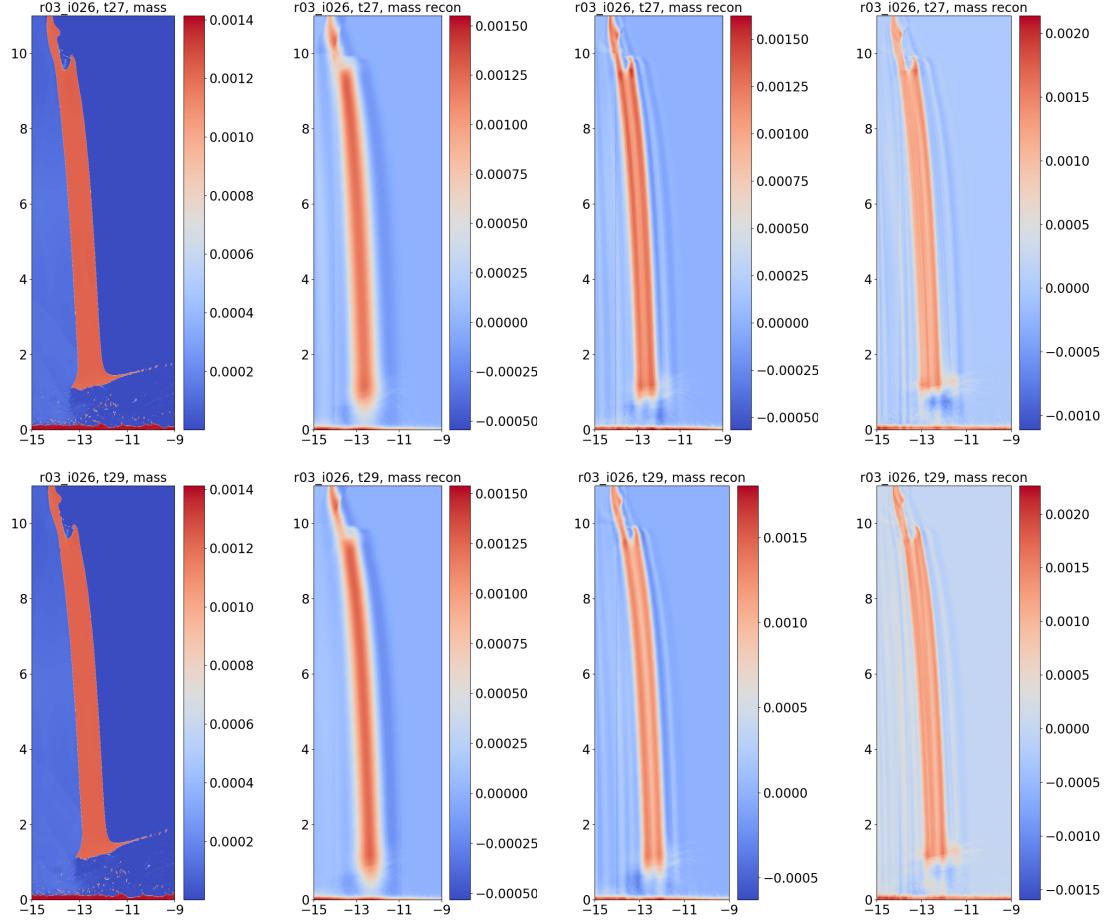
Figure 35: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i026** at time t27 (top) and t29 (bottom). The plate view shows from left to right: the original snapshot; followed by the reconstruction using 8, 20, and 47 weights.
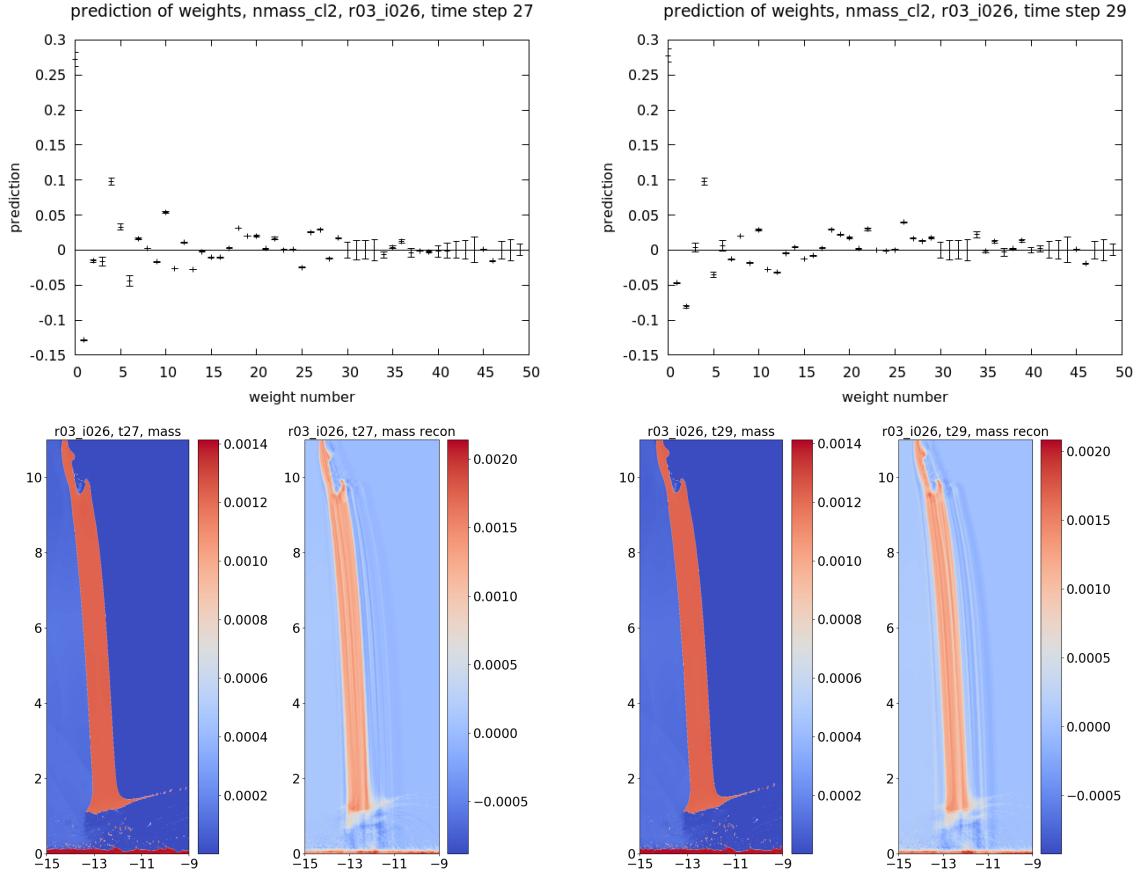
Figure 36: Reconstruction of the variable *mass*, after clustering, for test simulation **r03_i026** at time t27 (left) and t29 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 30 weights.
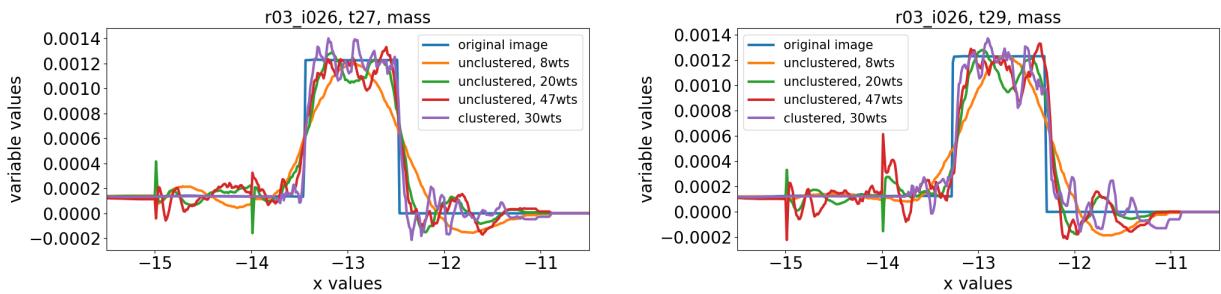
Figure 37: Comparison of the reconstruction of the variable *mass*, both with and without clustering, for test simulation **r03_i026** at time t27 (left) and t29 (right). The plots compare the value of the variable at $y = 6.0063$ for the different reconstructions to indicate how well they detect the plate boundary.
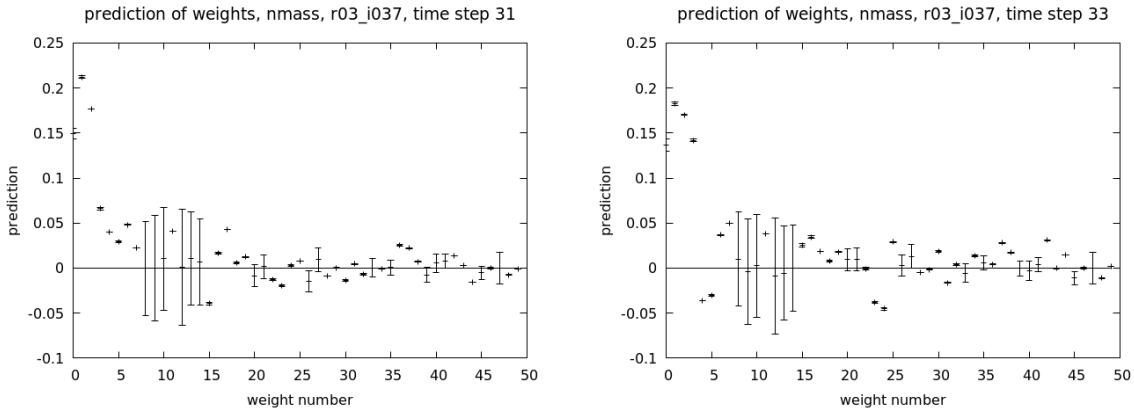
Figure 38: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i037** at time t31 (left) and t33 (right). The plots show the weight predictions (with 1 standard deviation error bar) for the first 50 weights.
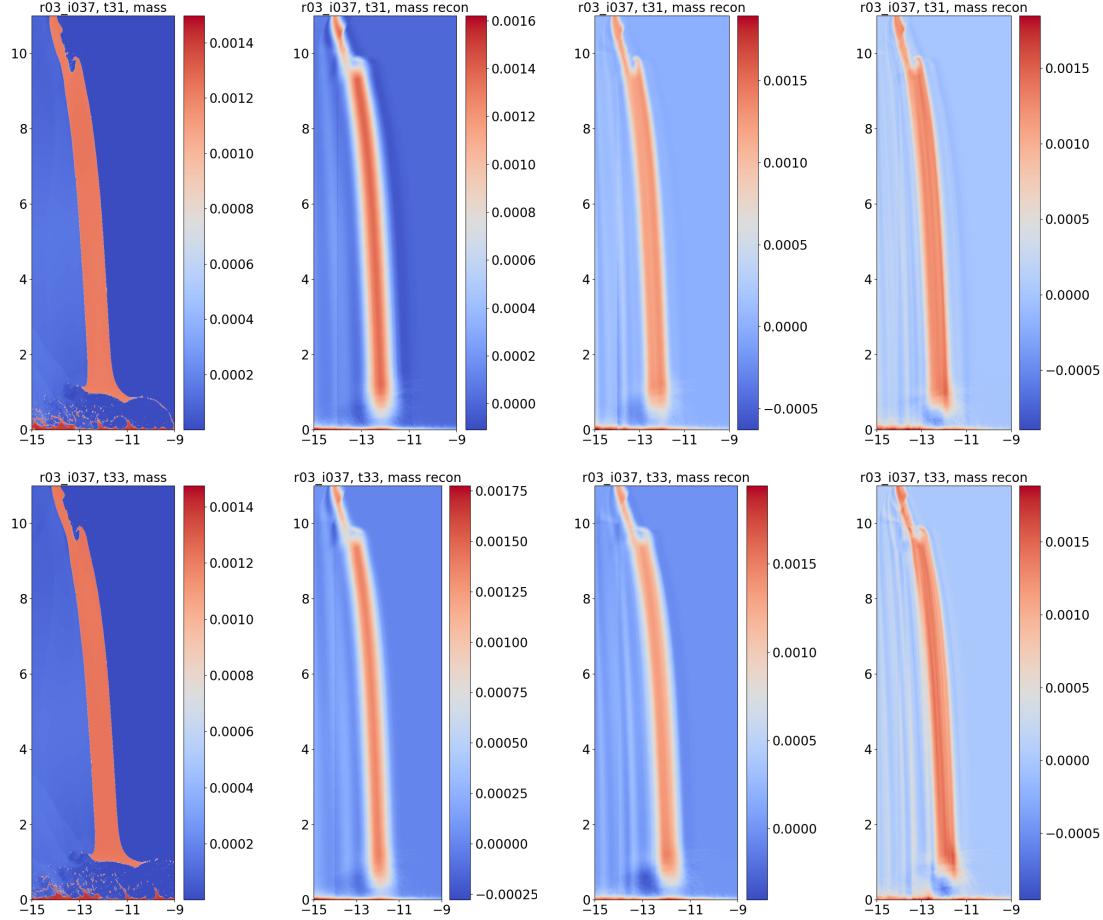

Figure 39: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i037** at time t31 (top) and t33 (bottom). The plate view shows from left to right: the original snapshot; followed by the reconstruction using 8, 20, and 47 weights.
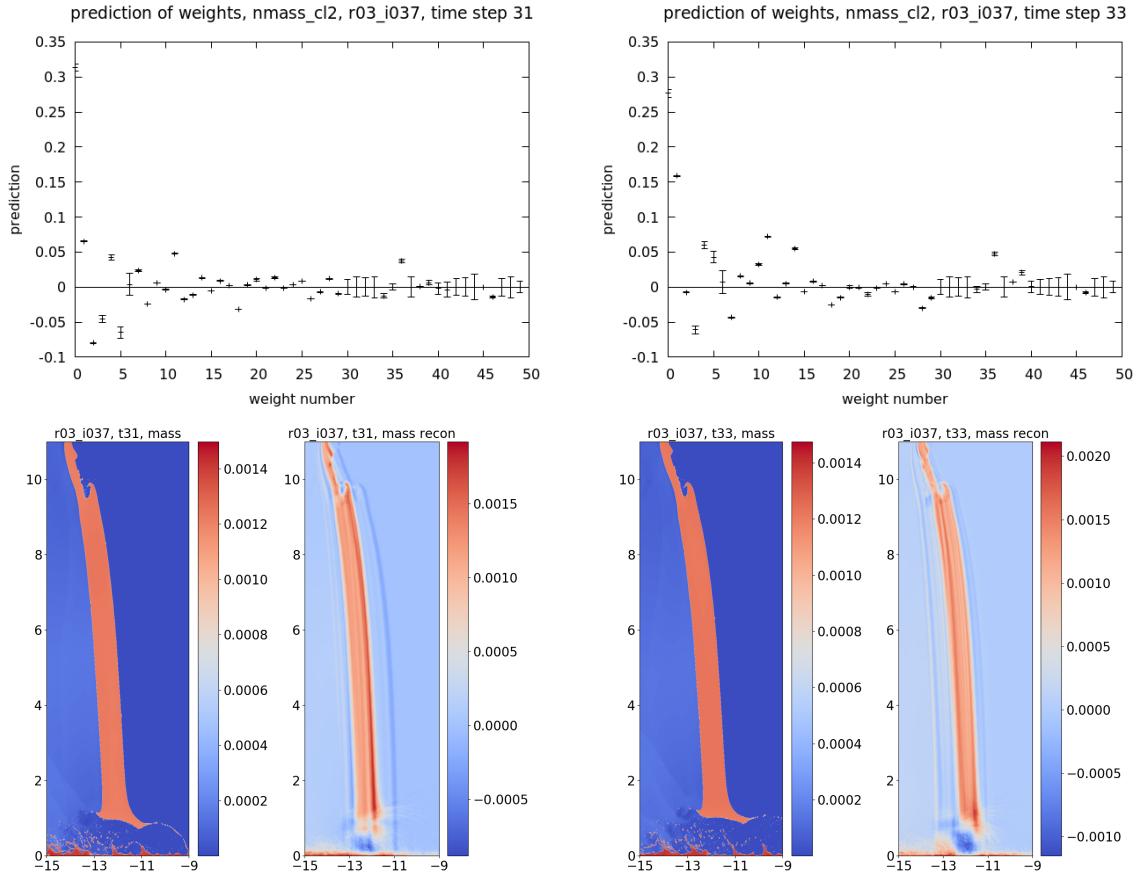
Figure 40: Reconstruction of the variable *mass*, after clustering for test simulation **r03_i037** at time t31 (left) and t33 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 30 weights.
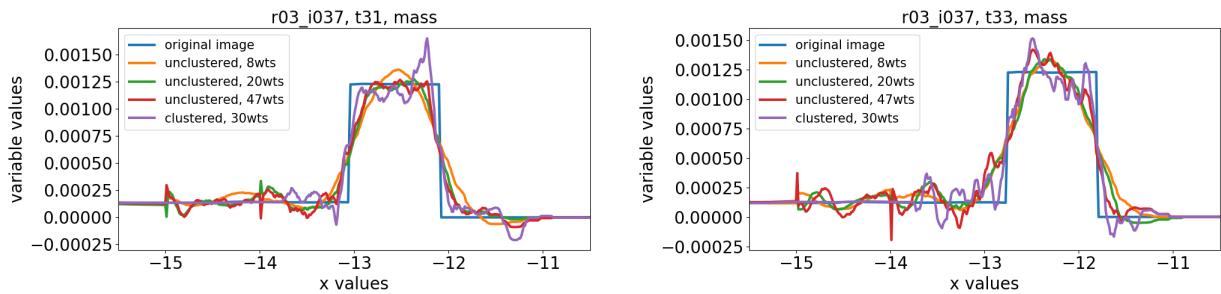

Figure 41: Comparison of the reconstruction of the variable *mass*, both with and without clustering, for test simulation **r03_i037** at time t31 (left) and t33 (right). The plots compare the value of the variable at $y = 6.0063$ for the different reconstructions to indicate how well they detect the plate boundary.
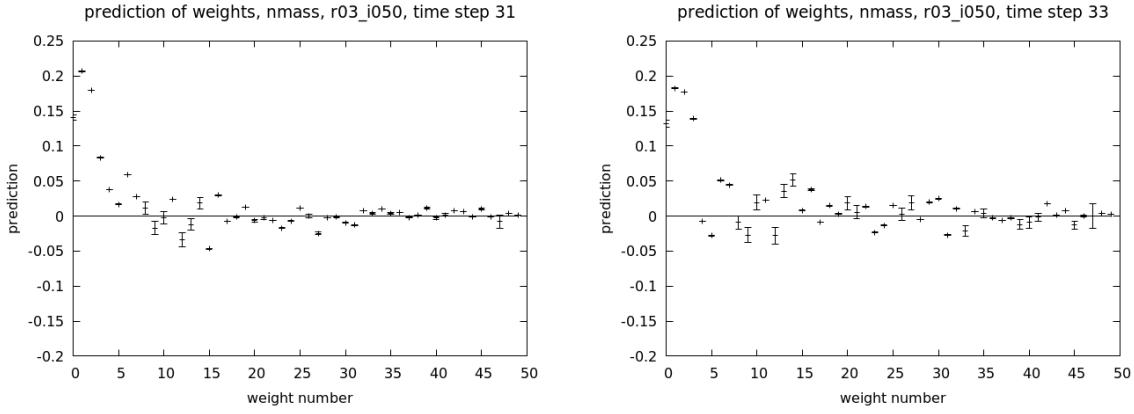
Figure 42: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i050** at time t31 (left) and t33 (right). Plots show the weight predictions (with 1 standard deviation error bar) for the first 50 weights.
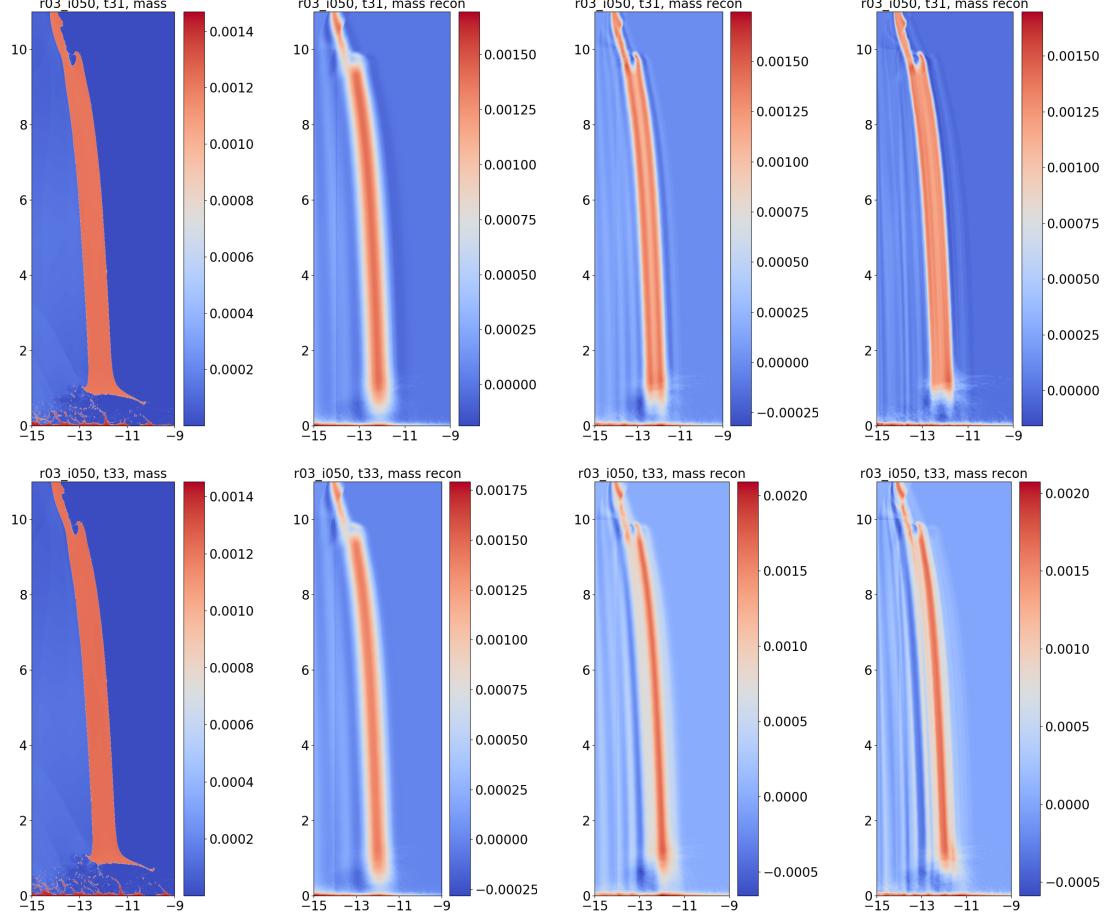
Figure 43: Reconstruction of the variable *mass*, without clustering, for test simulation **r03_i050** at time t31 (top) and t33 (bottom). The plate view shows from left to right: the original snapshot; followed by the reconstruction using 8, 20, and 47 weights.
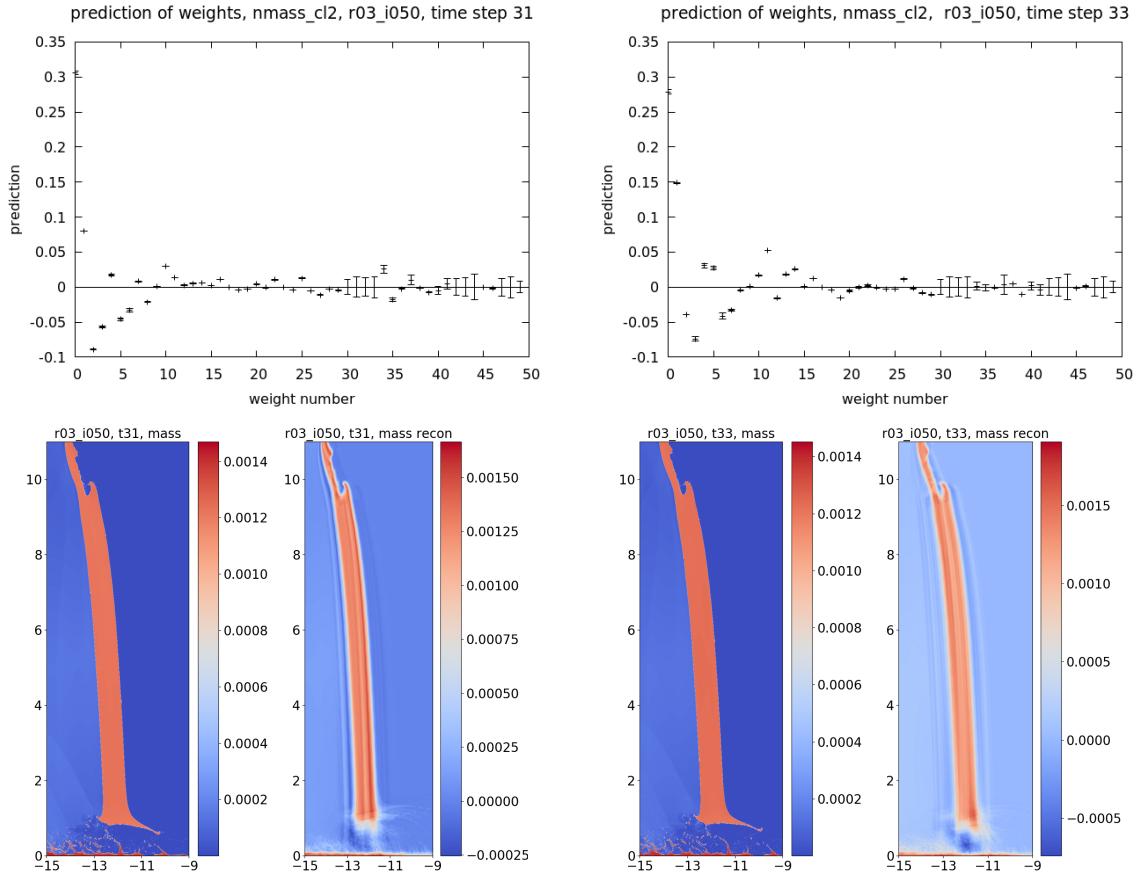
Figure 44: Reconstruction of the variable *mass*, after clustering, for test simulation **r03_i050** at time t31 (left) and t33 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 30 weights.
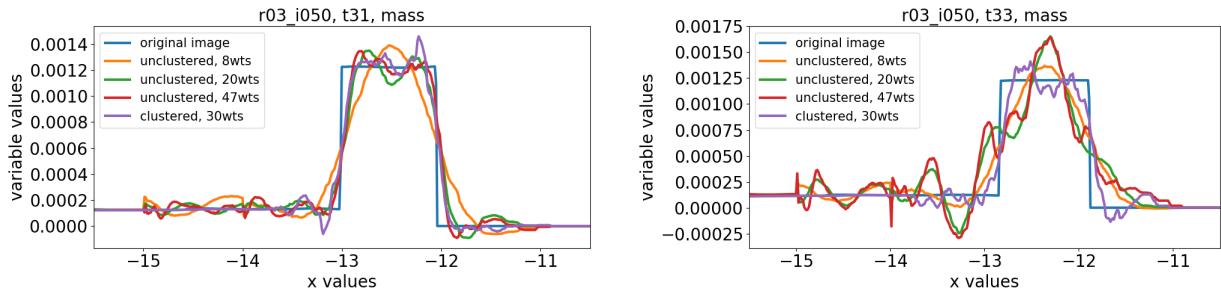

Figure 45: Comparison of the reconstruction of the variable *mass*, both with and without clustering, for test simulation **r03_i050** at time t31 (left) and t33 (right). The plots compare the value of the variable at $y = 6.0063$ for the different reconstructions to indicate how well they detect the plate boundary.

E   Appendix: Eigen-snapshots for x-momentum variable, with clustering
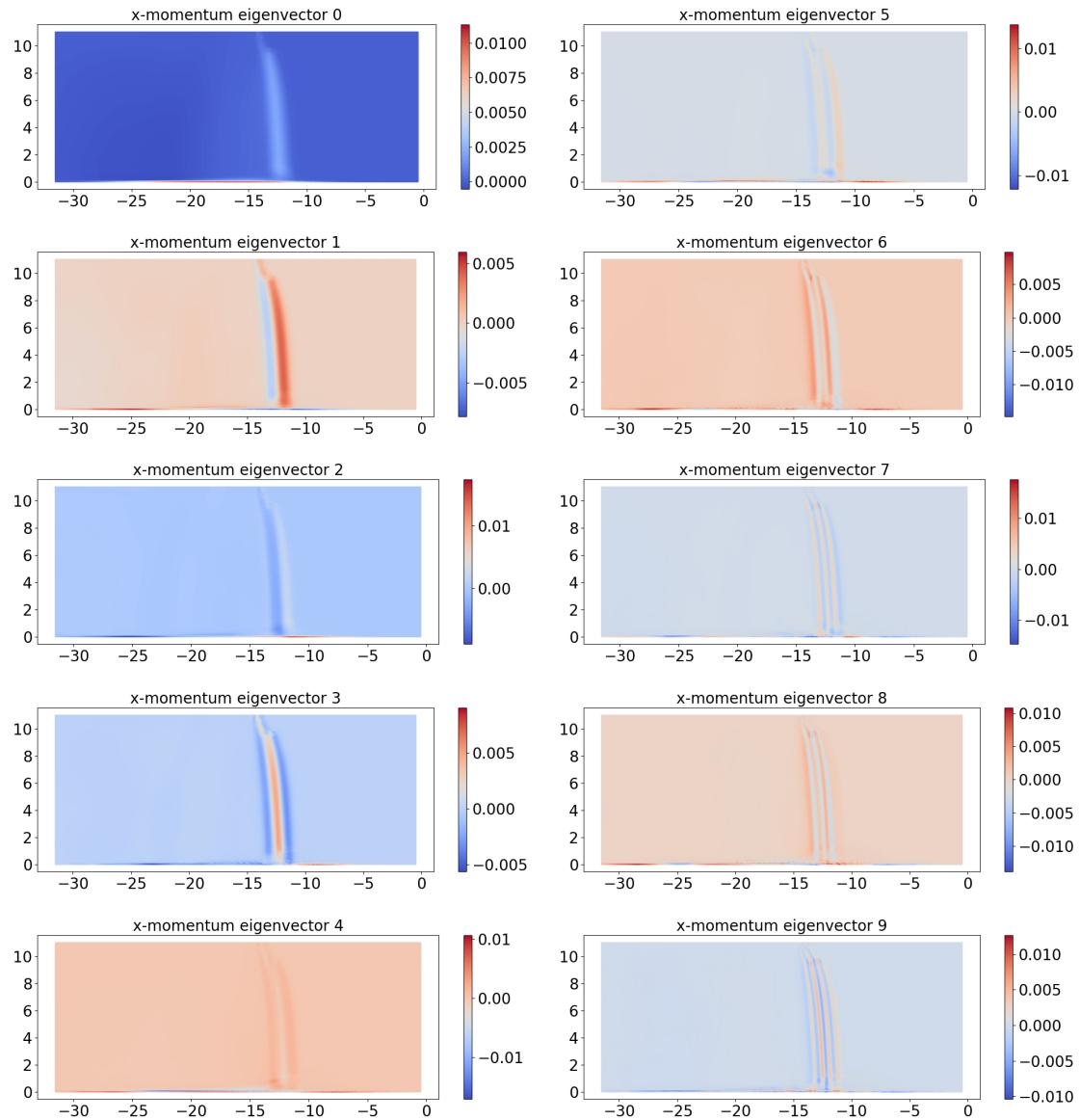
Figure 46: Eigen-snapshots (0-4 on left and 5-9 on right) for variable *x-momentum*, with clustering. The color bars are different across eigen-snapshots.

F Appendix: Reconstructed x-momentum (after clustering) for all seven test cases
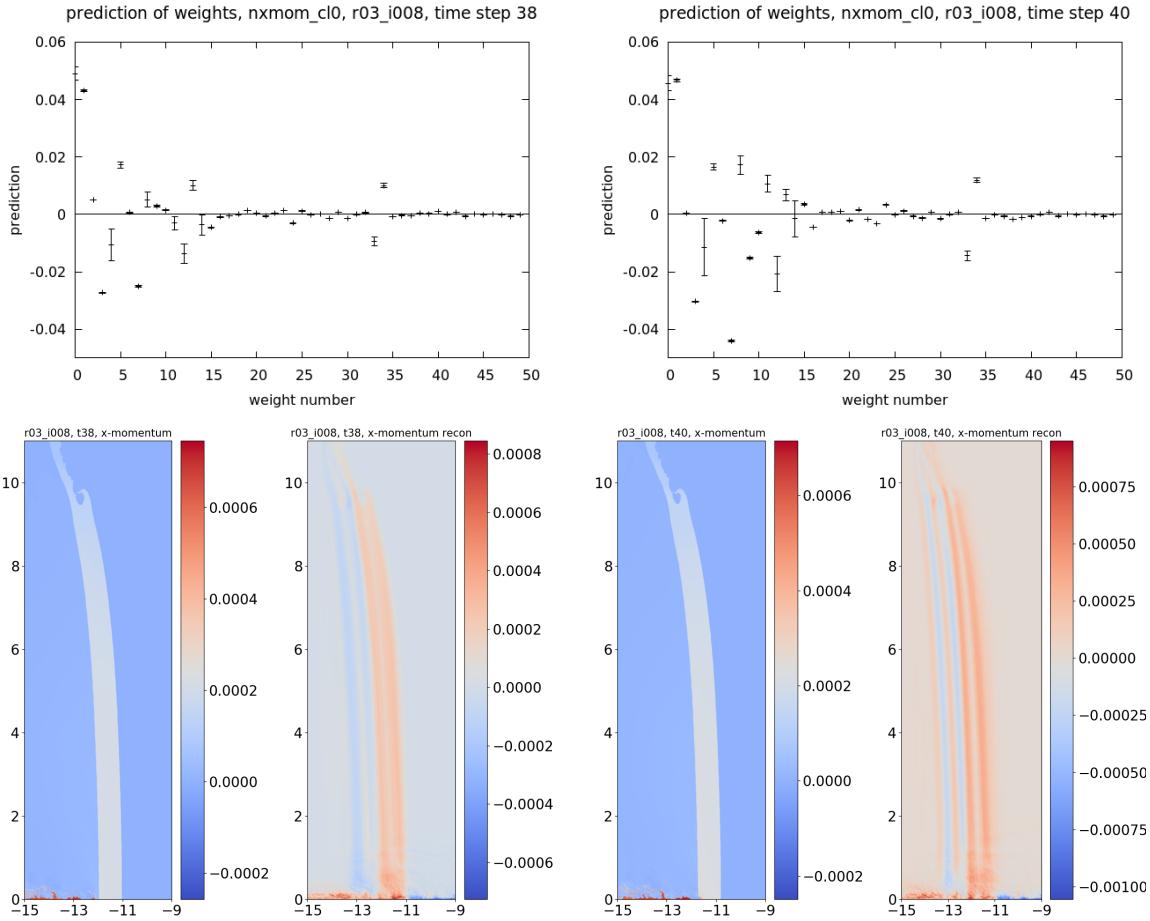
Figure 47: Reconstruction of the variable *x-momentum*, after clustering, for test simulation **r03_i008** at time t38 (left) and t40 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 50 weights. The color bars are different across images.
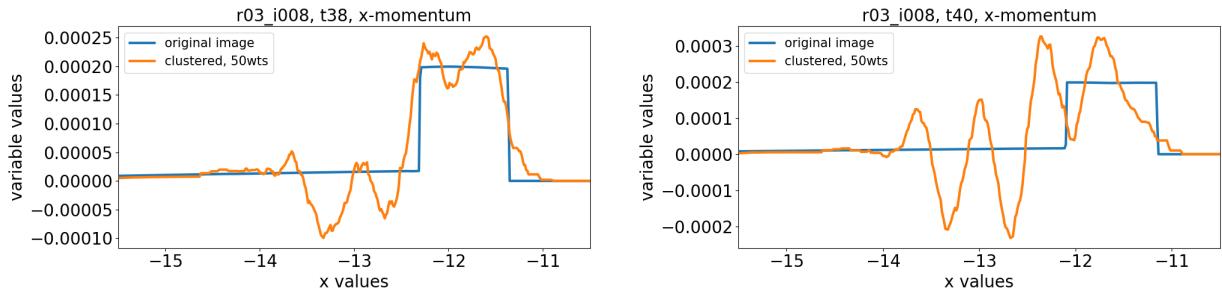

Figure 48: Comparison of the reconstruction of the variable *x-momentum*, with clustering, for test simulation **r03_i008** at time t38 (left) and t40 (right). The plots compare the value of the variable at $y = 6.0063$ for the reconstruction to indicate how well it detects the plate boundary.

52
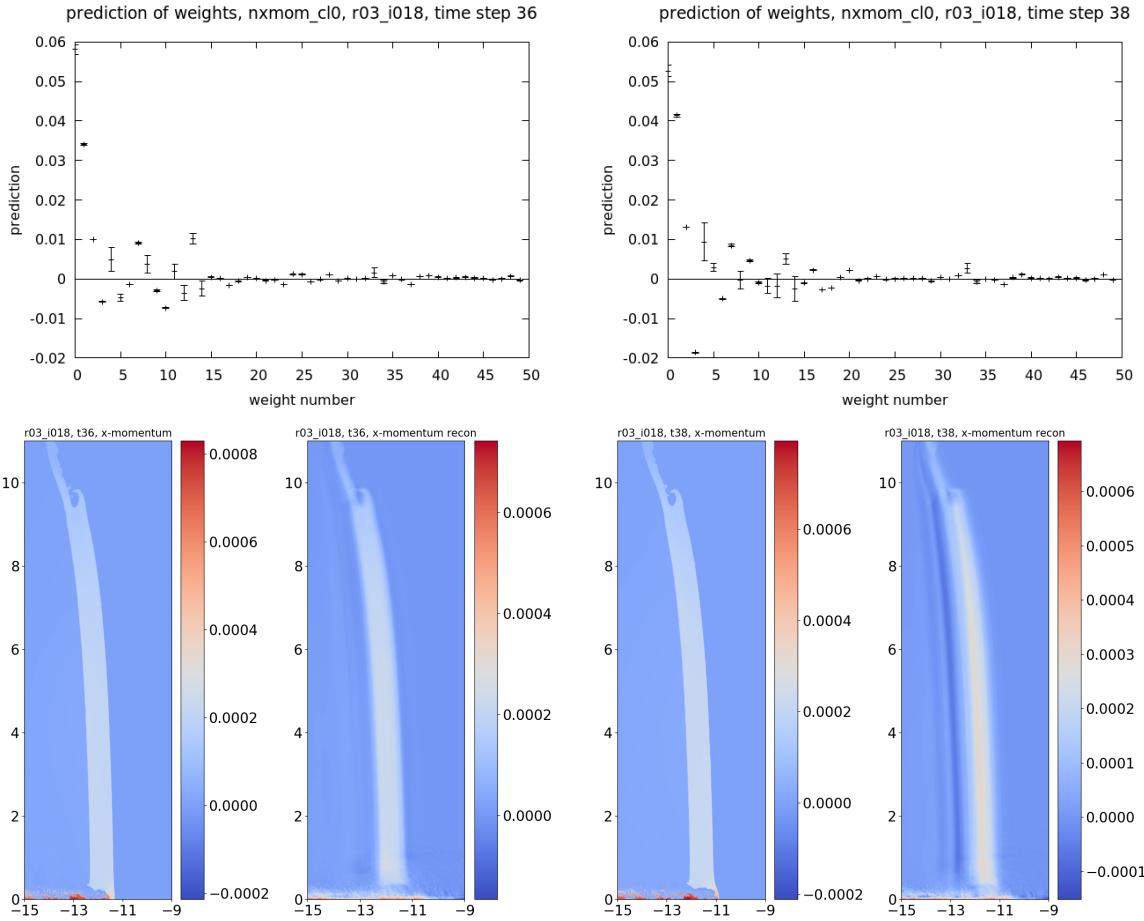
Figure 49: Reconstruction of the variable *x-momentum*, after clustering, for test simulation **r03_i018** at time t36 (left) and t38 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 50 weights. The color bars are different across images.
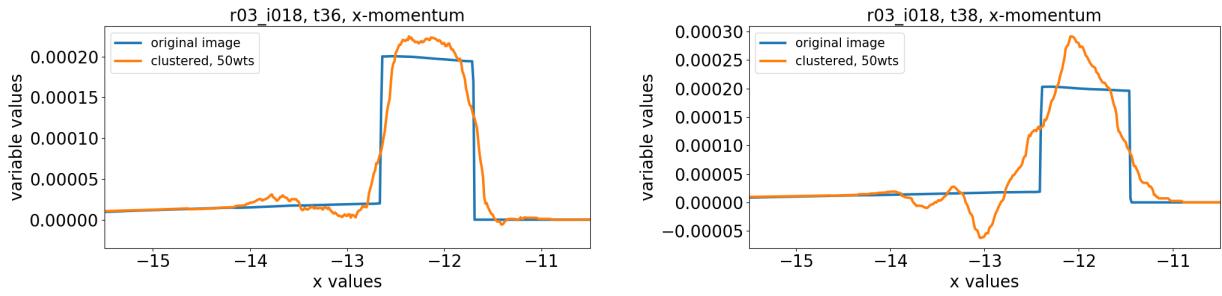

Figure 50: Comparison of the reconstruction of the variable *x-momentum*, with clustering, for test simulation **r03_i018** at time t36 (left) and t38 (right). The plots compare the value of the variable at $y = 6.0063$ for the reconstruction to indicate how well it detects the plate boundary.
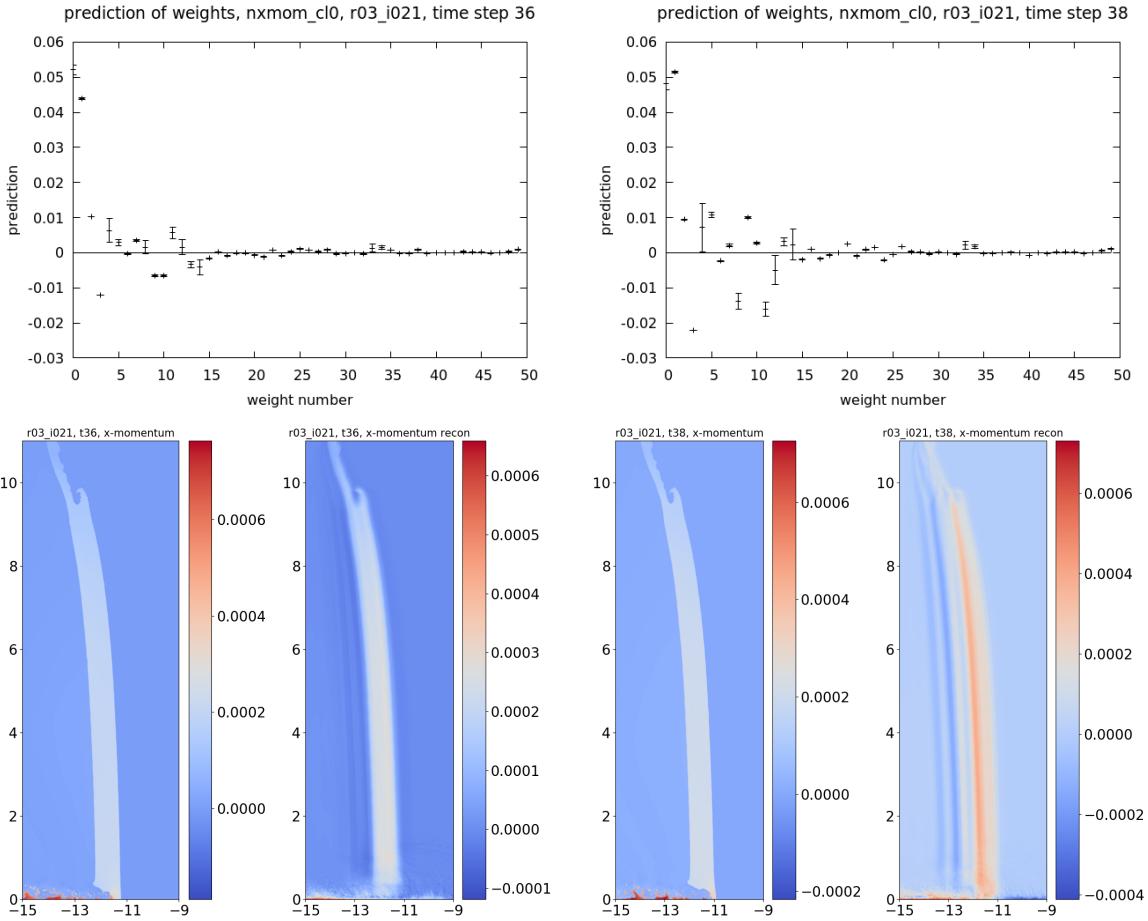
Figure 51: Reconstruction of the variable *x-momentum*, after clustering, for test simulation **r03_i021** at time t36 (left) and t38 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 50 weights. The color bars are different across images.
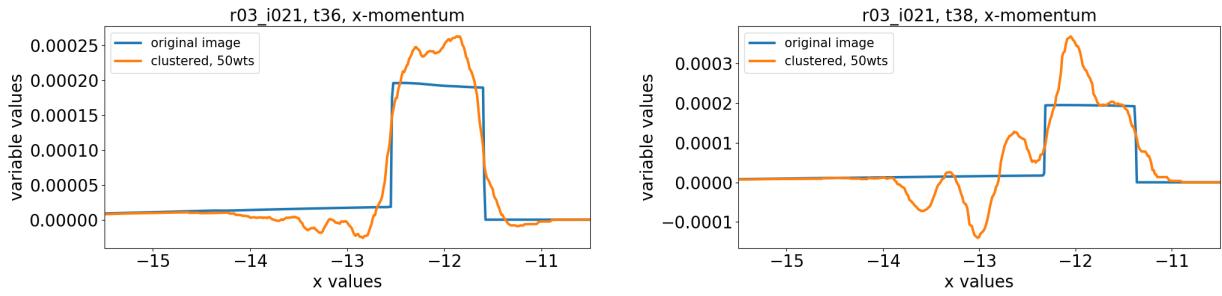

Figure 52: Comparison of the reconstruction of the variable *x-momentum*, with clustering, for test simulation **r03_i021** at time t36 (left) and t38 (right). The plots compare the value of the variable at $y = 6.0063$ for the reconstruction to indicate how well it detects the plate boundary.
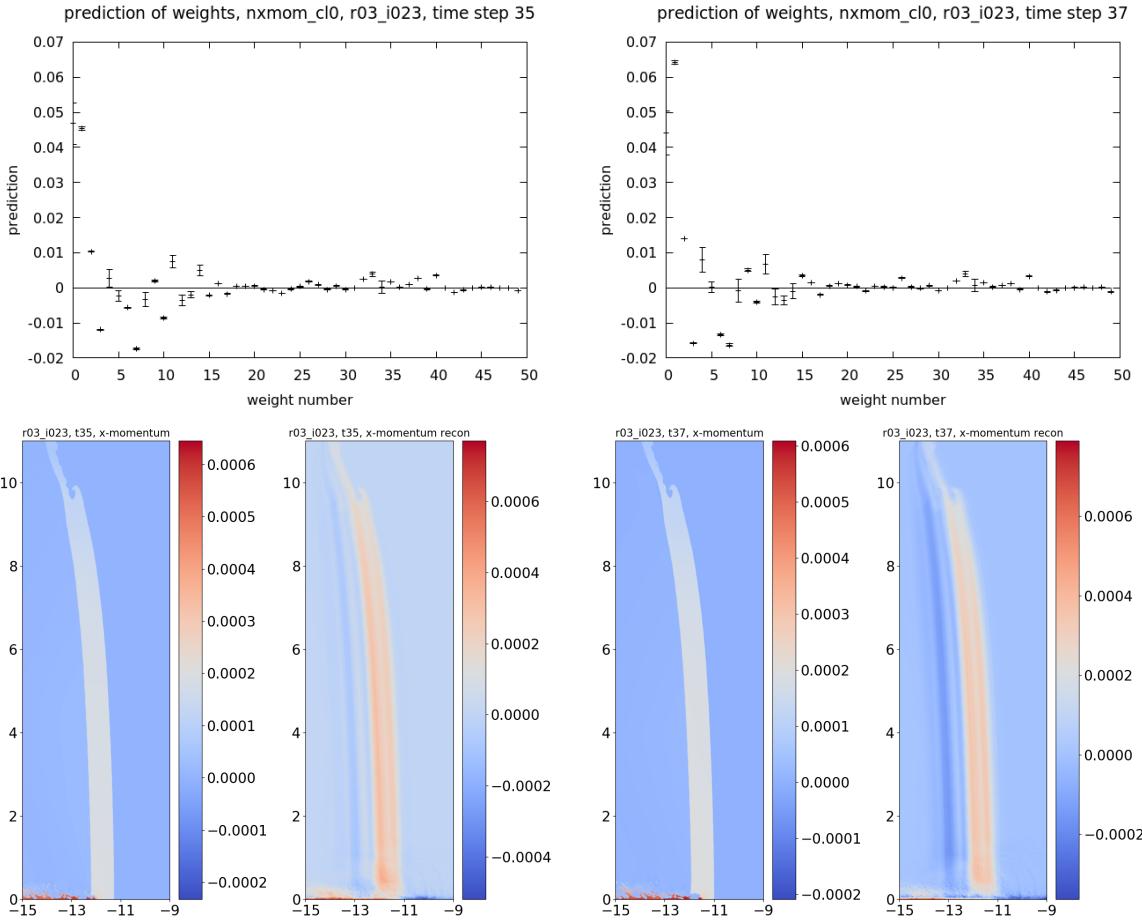
Figure 53: Reconstruction of the variable *x-momentum*, after clustering, for test simulation **r03_i023** at time t35 (left) and t37 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 50 weights. The color bars are different across images.
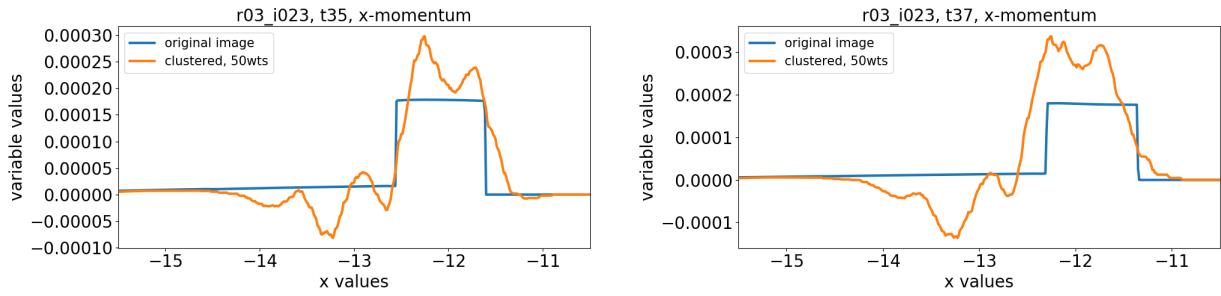

Figure 54: Comparison of the reconstruction of the variable *x-momentum*, with clustering, for test simulation **r03_i023** at time t35 (left) and t37 (right). The plots compare the value of the variable at $y = 6.0063$ for the reconstruction to indicate how well it detects the plate boundary.
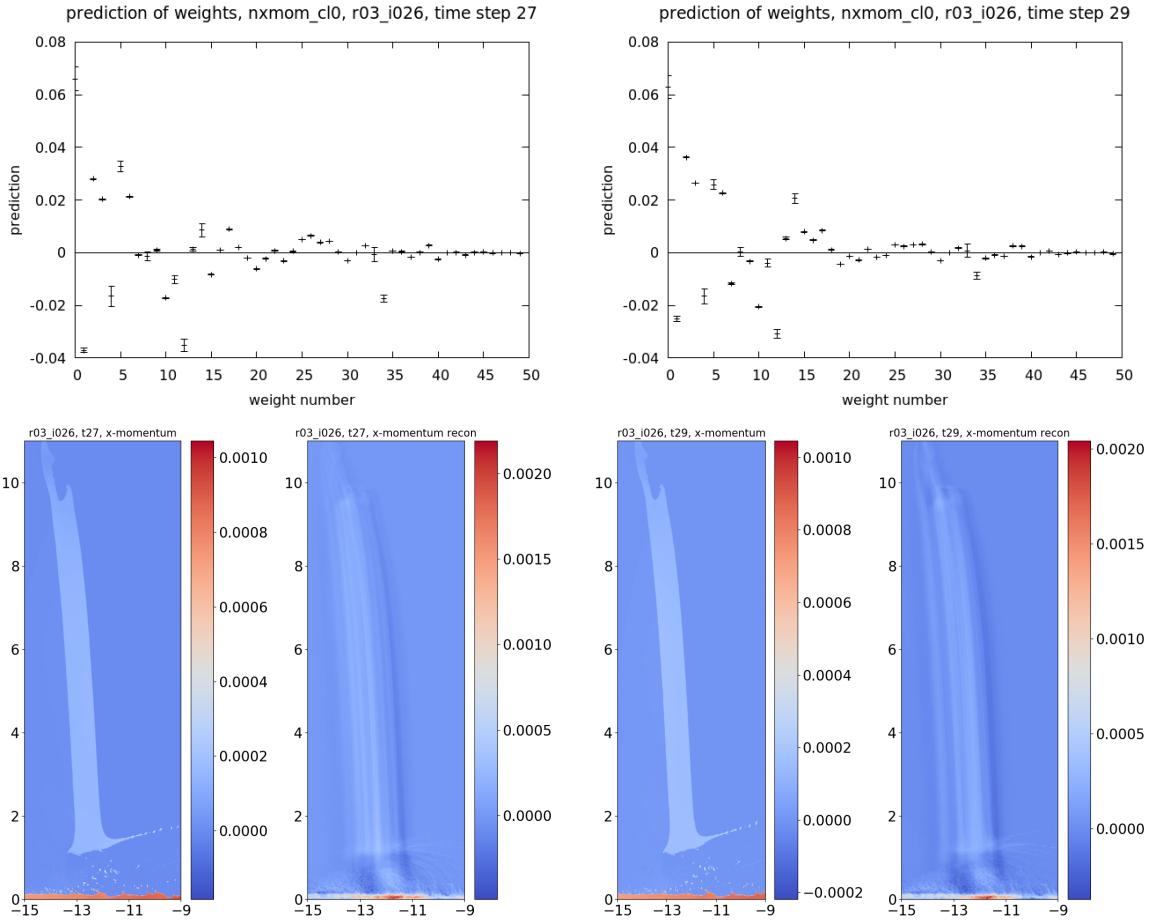
Figure 55: Reconstruction of the variable *x-momentum*, after clustering, for test simulation **r03_i026** at time t27 (left) and t29 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 50 weights. The color bars are different across images.
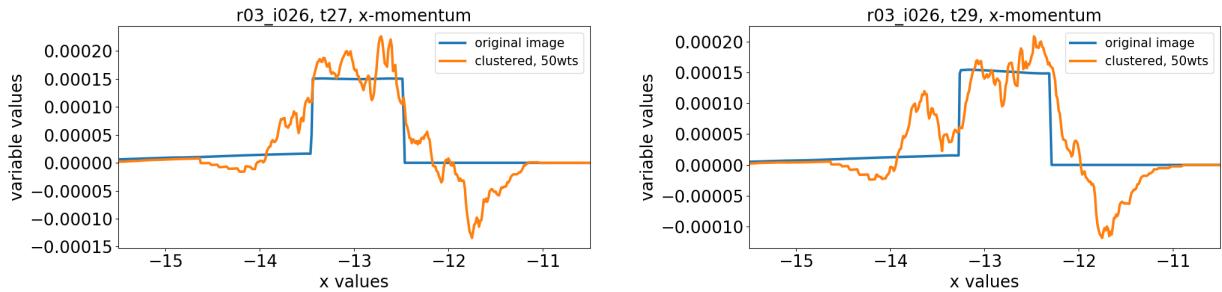
Figure 56: Comparison of the reconstruction of the variable *x-momentum*, with clustering, for test simulation **r03_i026** at time t27 (left) and t29 (right). The plots compare the value of the variable at $y = 6.0063$ for the reconstruction to indicate how well it detects the plate boundary.
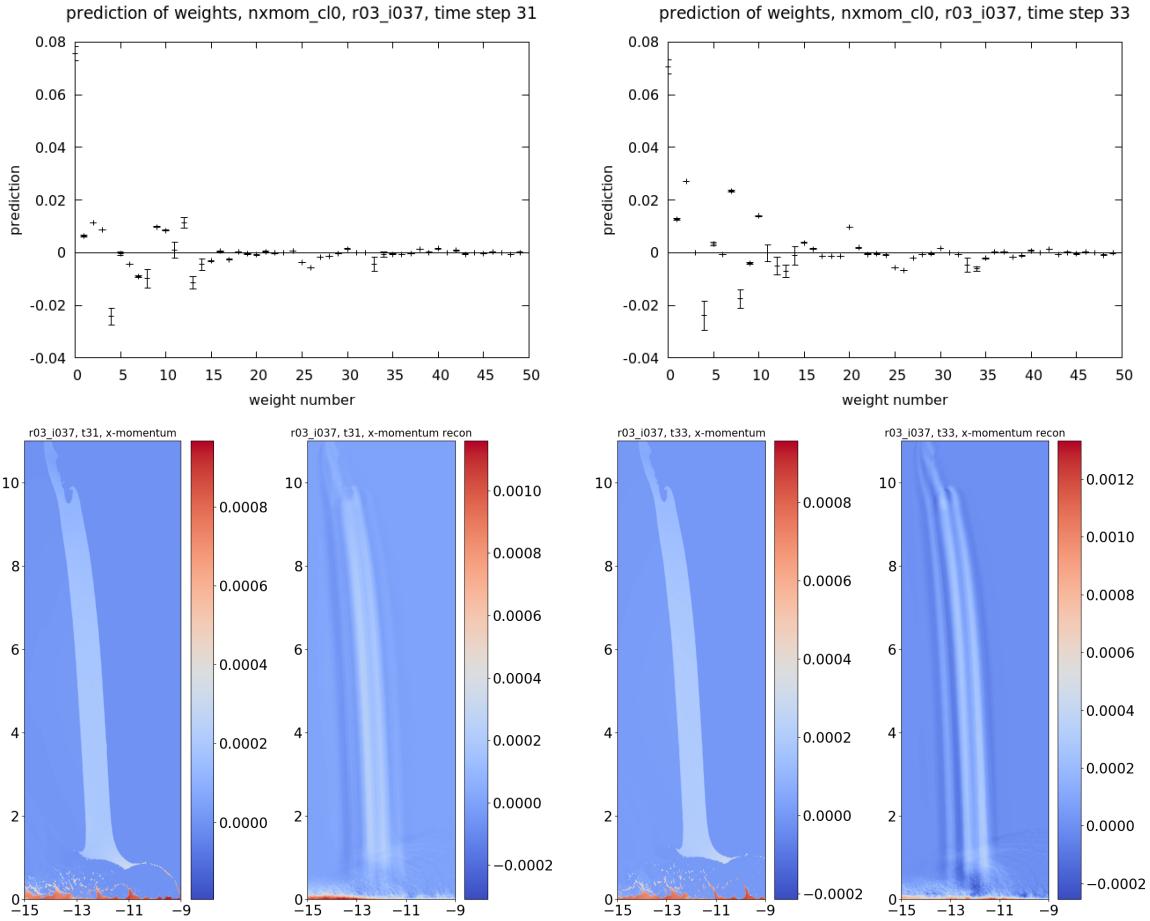
Figure 57: Reconstruction of the variable *x-momentum*, after clustering, for test simulation **r03_i037** at time t31 (left) and t33 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 50 weights. The color bars are different across images.
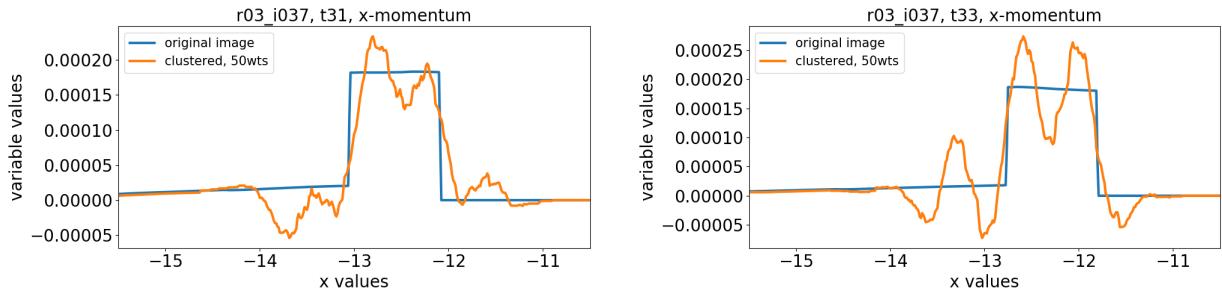

Figure 58: Comparison of the reconstruction of the variable *x-momentum*, with clustering, for test simulation **r03_i037** at time t31 (left) and t33 (right). The plots compare the value of the variable at $y = 6.0063$ for the reconstruction to indicate how well it detects the plate boundary.
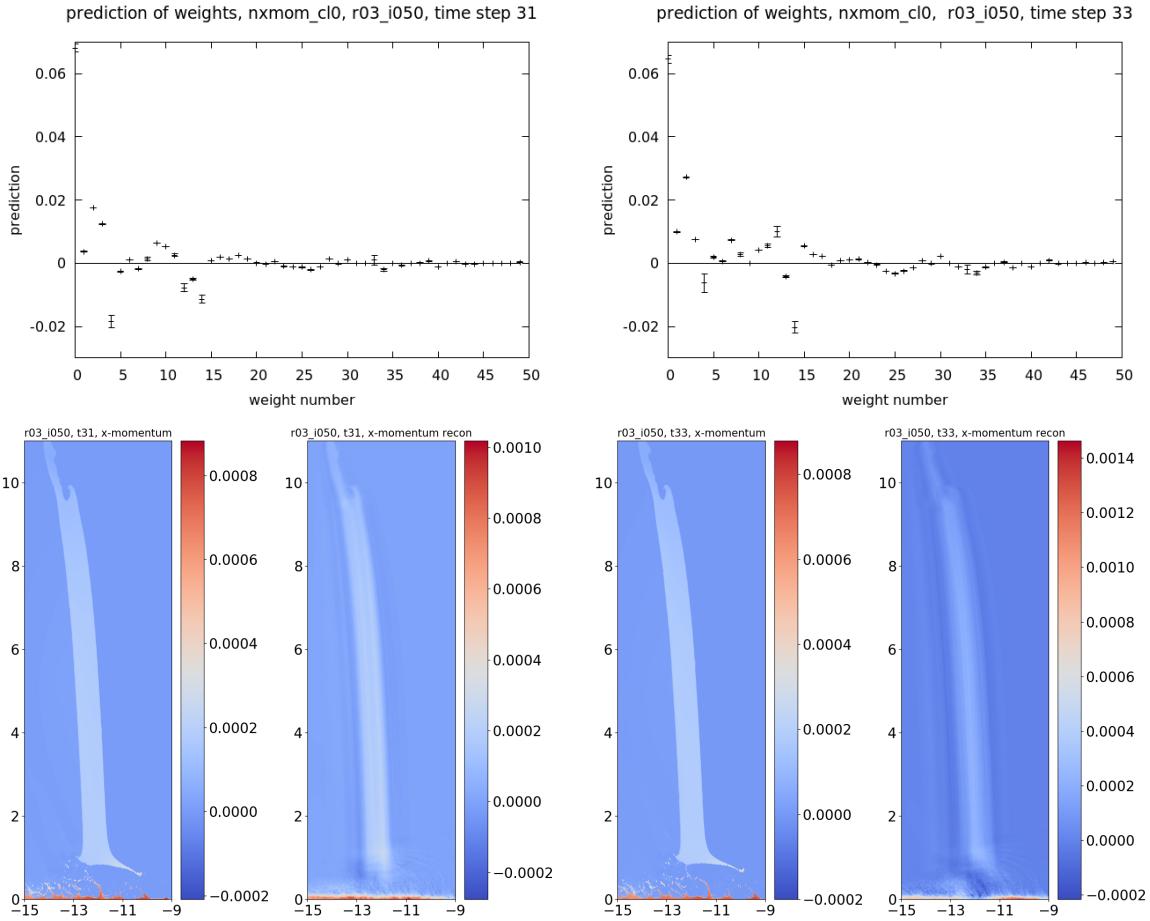
Figure 59: Reconstruction of the variable *x-momentum*, after clustering, for test simulation **r03_i050** at time t31 (left) and t33 (right). Top: the weight predictions (with 1 standard deviation error bar) for the first 50 weights. Bottom: each pair shows the plate view of the original snapshot and the reconstruction using 50 weights. The color bars are different across images.
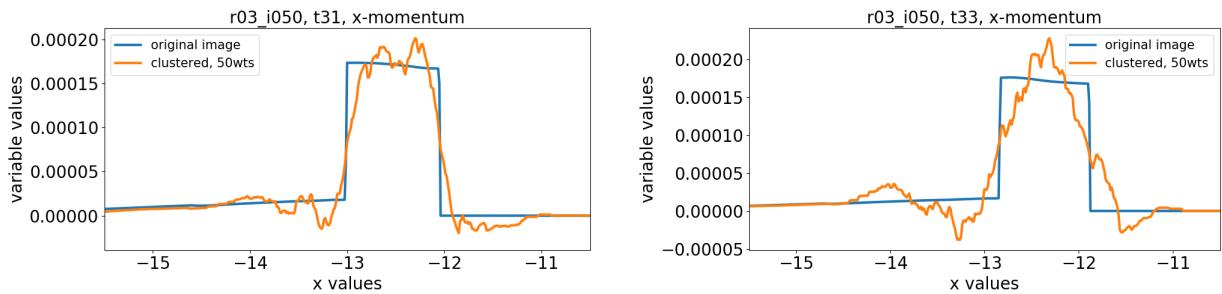

Figure 60: Comparison of the reconstruction of the variable *x-momentum*, with clustering, for test simulation **r03_i050** at time t31 (left) and t33 (right). The plots compare the value of the variable at $y = 6.0063$ for the reconstruction to indicate how well it detects the plate boundary.

58