Politecnico di Torino

SCIENCES SORBONNE UNIVERSITÉ

Master of Science Thesis

Department of Applied Sciences and Technology

Master Degree in
Physics of Complex Systems

Enhancing blood clot simulations by Deep Learning and Model Order Reduction techniques

Candidate:

Alessandro Longhi

Supervisors:

Didier Lucor, *LISN laboratoire*
Amélie Fau, *ENS Paris-Saclay*
Rodrigo Rojano, *Cornell University*
Andrea Gamba, *Politecnico di Torino*

LISN LABORATOIRE INTERDISCIPLINAIRE DES SCIENCES DU NUMÉRIQUE

école normale supérieure paris-saclay

université PARIS-SACLAY

October 25th, 2022

Necessity, weight, and value are three concepts inextricably bound: only necessity is heavy, and only what is heavy has value ∼ M. Kundera

Acknowledgements

I start with a sincere thank you to my advisors Didier, Amélie and Rodrigo. You helped me going through my thesis work with helpful suggestions and enthusiasm, never judging but always supporting and giving me confidence. It has been a great first research experience.

I will always be grateful to my family, for blindly believing in me (sometimes a bit too much) and supporting psychologically and economically every decision I have made. These are two things I have always taken for granted, but they are not.

To the new friends from PCS and from the semester in Paris, thank you for all the time spent together, you made the (insane) Parisian semester unforgettable and worth experiencing.

To my old friends from home, thank you for always being present despite the long absences and all changes.

To Sofia Sofi, thank you for being the best person I know. Thank you for your (deep) lightness and optimism, I could not wish for someone better at my side.

Abstract

Thrombus formation in blood vessels is a medical problem which manifests itself both spontaneously and after the insertion of medical devices inside human bodies. Around 500000 people die in the EU each year because of it, it is thus crucial to develop robust computational methods to predict its formation. This work shows how Deep Learning and Model Order Reduction techniques can be applied in order to enhance physical simulations of blood clots formation. A few simulations were carried out with the software OpenFoam, to solve the Navier-Stokes equations for the blood flow and a set of parametrized PDEs, which couple the concentrations of the biochemical species with the velocity field of blood. Our aim is finding a method which allows to predict the evolution in time and space of some selected biochemical species giving as input **the parameters** on which the PDEs depend, **without** solving numerically the biochemical and mechanics equations. At this regard, we show how Proper Orthogonal Decomposition (POD) and a Neural Network (NN) architecture that combines a Convolutional Autoencoder (CAE) and a Deep Feed-Forward Neural Network (DFNN) can be exploited to give a good approximation of the PDEs solution. We then analyze some limitations of the method, such as the difficulty at making good predictions in time when the blood flow changes rapidly, and we propose some modifications to overcome these drawbacks.

Contents

1 Thrombus Simulations

1.1 Introduction and objectives

Thrombosis is an hemostatic process characterized by the formation of blood clots in a localized point within a blood vessel [1], preventing blood from circulating normally. A significant number of thrombosis cases happens when patients are treated with medical devices which are in contact with blood. This is due to the fact that artificial surfaces lack the anti-thrombotic properties of the endothelium [2]. Clot formation in implantable devices can lead to device failure or, if the clot breaks off, result in neurological or pulmonary damage (ischemic stroke). Better devices with improved blood compatibility are needed to reduce the incidence of adverse events. Computer simulations are a great tool to design the structure of these medical devices without conducting experiments. The simulations are used to identify and suppress blood stagnation zones or non-physiological forces that promote platelet activation and thrombosis. This work is based on computational fluid dynamic (CFD) simulations of thrombus formation inside a medical device.

Computational fluid dynamic methods are extremely costly in terms of time, and this poses a big limitation to the use of numerical simulations to test the efficacy of medical devices. More specifically, there is a **multi-scale** problem: the order of magnitude of the time step required to solve unsteady blood flows is 10 microseconds while the thrombus growth might take up to 6 hours. When using a time step of 10 microseconds thrombosis simulations cannot be used to optimize devices, since the simple fact of doing one accurate simulation considering unsteady flows is in itself a major achievement. Furthermore, current thrombosis simulations can involve up to one hundred biochemical species, and for each species a supplemental Convection-Diffusion-Reaction (CDR) equation must be solved, increasing the computational cost of the simulation. This problem is amplified (in time terms) when there is the need to test what happens by changing the parameters on which the equations describing the model depend, since in principle the simulations should be run for every set of parameter. In this work we will present a method which makes it possible to **avoid performing a simulation** for every parameter instance.

In the current study, the thrombosis simulations of [3] were used to develop a model order reduction [4] through deep learning methods. The medical device is used for the simulations is the platelet function analyzer PFA-100®, a coagulation testing device used to asses the primary hemostasis response [5], useful for screening for von Willebrand's disease. The quantities of interest which are given by the simulations are the biochemical species present in Table 1, whose concentrations are coupled with the blood flow. Obtaining the values of those species in time and space, is crucial to understand if and how the thrombus will grow. Some of them are more important for the creation of the thrombus, such as the activated platelets (AP) and the von Willebrand factor (vWF). For example the presence of the thrombus would affect the concentration of AP, making its concentration decrease where the thrombus is growing. vWF instead unfolds in response to strong flow gradients and facilitates rapid recruitment of platelets in flowing blood [6]. Two states of vWF are considered: collapsed vWF_c and stretched vWF_s.

Although the final goal of this project is being able to predict with high accuracy

Species	Concentration
RP	216×10^3 PltμL^{-1}
AP	2.16×10^3 PltμL^{-1}
$\mathbf{vWF_c}$	1000 nmol m^{-3}
PT	1.1×10^6 nmol m^{-3}
TB	0 nmol m^{-3}
AT	2.844×10^6 nmol m^{-3}
ADP	0 nmol m^{-3}
$\mathbf{TxA_2}$	0 nmol $m-3$
$\mathbf{AP}_d$	0 nmol m^{-3}
$\mathbf{RP}_d$	0 nmol m^{-3}
$\mathbf{vWF_s}$	0 nmol m^{-3}

Table 1: Biochemical species and their inlet concentrations.

the blood flow and the change in time and space of the biochemical species in order to predict the growth of the thrombus, what will be shown here regards the time interval just before the thrombus formation. The reason for this choice is to ensure that the computational and mathematical tools used work, and thus a simpler problem is tackled at first: we will focus on developing tools to get good predictions of the biochemical species in time and space before the thrombus formation, given the parameters of the model as input. The computational methods, the biological considerations and the model presented can be found in [3].

1.2 Geometry

In Figure 1 a cross-section of PFA-100®is presented, whose main component is a bio-active membrane with a central orifice of 147 microns diameter. The blood is aspirated through a capillary towards the membrane, and as blood flows through the central orifice of the membrane, a thrombus forms until occlusion is achieved. The membrane is coated with collagen and epinephrine or Adenosine diphosphate (ADP) to promote thrombus formation. When the orifice is fully occluded a Closure Time (CT) is obtained which is the quantity used to diagnose patients. PFA-100®has a cylindrical shape, with a smaller diameter D_c for the capillary and a bigger diameter D_r for the reservoir. The clot is expected to form in between the two membranes. Because of the cylindrical symmetry, we reduce the $3D$ problem to a $2D$ problem. Moreover, we assume that taking only one side of the axis of symmetry does not affect the result. In Figures 3 and 4 the shape of the 'reduced' geometry used in the simulation is shown, along with the concentration of vWFs and AP at 3 different seconds for a given set of the PDEs parameters. The computational mesh is composed of 68650 hexahedral cells with a finest resolution of $\Delta x = 3\mu m$ located in the membrane orifice. The mesh is non uniform, meaning that the space between cell points becomes smaller in the proximity of the membrane orifice. A dual time step method is used to improve the computational cost of the simulations. A time step of $\Delta t_{CDR} = 1 \times 10^{-3}$ s is used to solve the species equations and the thrombus growth, while the blood flow equations are solved with a time step $\Delta t_{CFD} = \Delta t_{CDR}/r_{flow} =$

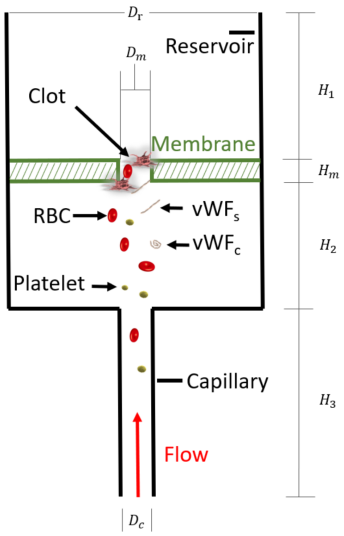

Figure 1: Cross section of PFA-100® testing cartridge showing the capillary, the central membrane with orifice, and reservoir. As whole blood is aspired through the cartridge blood constituents aggregate in the coated membrane orifice. The dimensions of the PFA-100®cartridge as considered in the simulations are: $D_c = 200\mu$m, $D_m = 147\mu$m, $D_r = 1200\mu$m, $H_1 = 1000\mu$m, $H_2 = 400\mu$m, $H_3 = 1200\mu$m, and $H_m = 200\mu$m. Image from [3].

1×10^{-8} s, to ensure the Courant–Friedrichs–Levy stability condition [7].

1.3 Mathematical model

The mathematical model behind the simulations aims at describing two coupled phenomena: the blood flow and the evolution in time and space of the biochemical species of Table 1. The pressure and velocity fields p and $\boldsymbol{u}$ are obtained by solving the equations of conservation of mass and linear momentum:

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\rho(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \mu \nabla^2 \mathbf{u} - \frac{C_2}{(1-\phi)} f(\phi)\mathbf{u} \tag{2}$$

where μ is the asymptotic dynamic viscosity of blood and ρ is the density. The scalar field ϕ represents the volume fraction of thrombus which comprises deposited platelets and fibrin. To avoid a singularity in the source term, the denominator is set as $(1-\phi) \in [\epsilon, 1]$, where ϵ is a small number. C_2 is the hindrance constant introduced by Wu et al. [8], which assumes that the thrombus is composed of densely compact spherical particles. $f(\phi)$ is the hindrance function. The biochemical species are modeled using a set of Convection-Diffusion-Reaction (CDR) equations to quantify their spatial and temporal dynamics:

$$\frac{\partial c_i}{\partial t} = \nabla \cdot (D_i \nabla c_i) - \mathbf{v}_f \cdot \nabla c_i + r_i \tag{3}$$

with c_i the concentration of species i, D_i the diffusion coefficient, $\mathbf{v}_f$ the velocity vector field and r_i the reaction source term that accounts for biochemical interac-

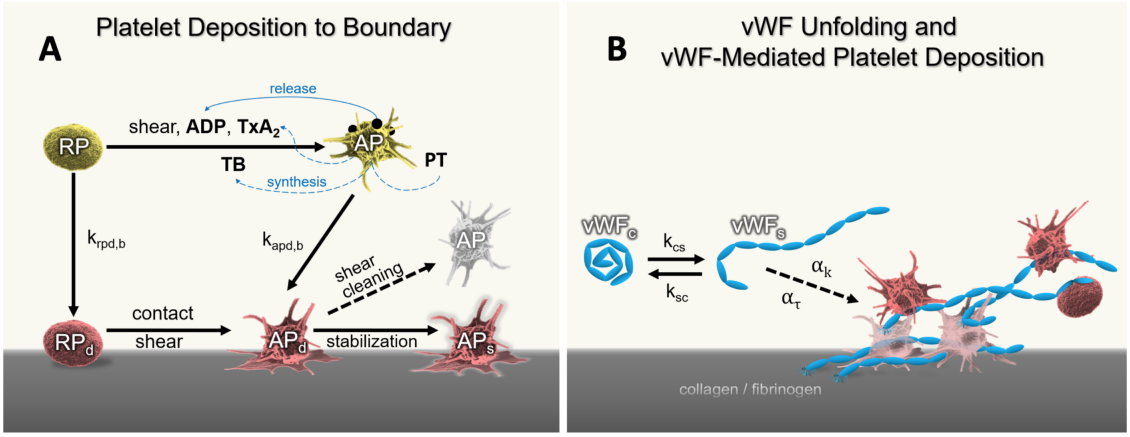

Figure 2: A)Diagram of platelet activation and deposition in the thrombosis model. $k_{rpd,b}$ and $k_{apd,b}$ are the rate for the deposition to the surface of resting and activated platelets. Mechanical shear or the combination of ADP, TxA_2 and thrombin can activate resting platelets. B)Representation of vWF unfolding and vWF-mediated platelet deposition and aggregation. Stretched vWFs amplify the deposition rate of free-flowing platelets by a_k and increase the resistance of deposited platelets to shear cleaning by a_τ. Image from [3].

tions. Figure 2 shows a schematic representation of vWFs unfolding due to high shear rate , which causes an increase in platelet deposition.

These parametrized CDR equations are the ones that depend on the parameters that we want to vary, and later a method which combines Deep Learning and Model Order Reduction will be described to avoid solving these PDEs numerically for a given parameter instance. The full list of parameters on which the CDR equations depend can be found in [3]. In this work we considered only the 5 most influential parameters shown in Table 2, in order to face a simpler problem. For a more detailed analysis on how the parameters of the model enter in the CDR equations through the reaction source term we refer to [8]. Before running the thrombosis simulations, a steady blood flow was obtained to improve stability at the first instants of the thrombosis simulation. This means that every new simulation has a new set of parameters $\boldsymbol{\mu}$ that leads to a different concentration in time and space of the biochemical species, not of the blood flow. However, the way the concentrations evolve affects the way the thrombus grows. This ultimately will affect also the blood flow. Thus the parameters change acts indirectly on the blood flow. Notice that we have

Parameter	Definition	Interval
$\dot{\gamma}_{vWF}$	Critical shear rate for vWF unfolding	$[3300 \quad 7700]\ s^{-1}$
t_{vWF}	vWF relaxation time	$[0.03 \quad 0.07]\ s$
$Wi_{eff,hyst}$	Wi_{eff} hysteresis value	$[0.1896 \quad 0.4424]$ dimensionless
vWF_s^{crit}	Critical vWF_S concentration value	$[30 \quad 70]\ m^{-3}$
t_{act}	Platelet characteristic activation time	$[0.05 \quad 0.5]\ s$

Table 2: Parameters varied in the simulations (uniformly distributed).

two sets of quantities: the **parameters** of Table 2 and the **biochemical species**

of Table 1, that are not to be confused: the former are the parameters of the CDR equations (3) that we want to vary, the latter are the fields of interest that we want to predict at the variation of those parameters, **without** running the simulations.

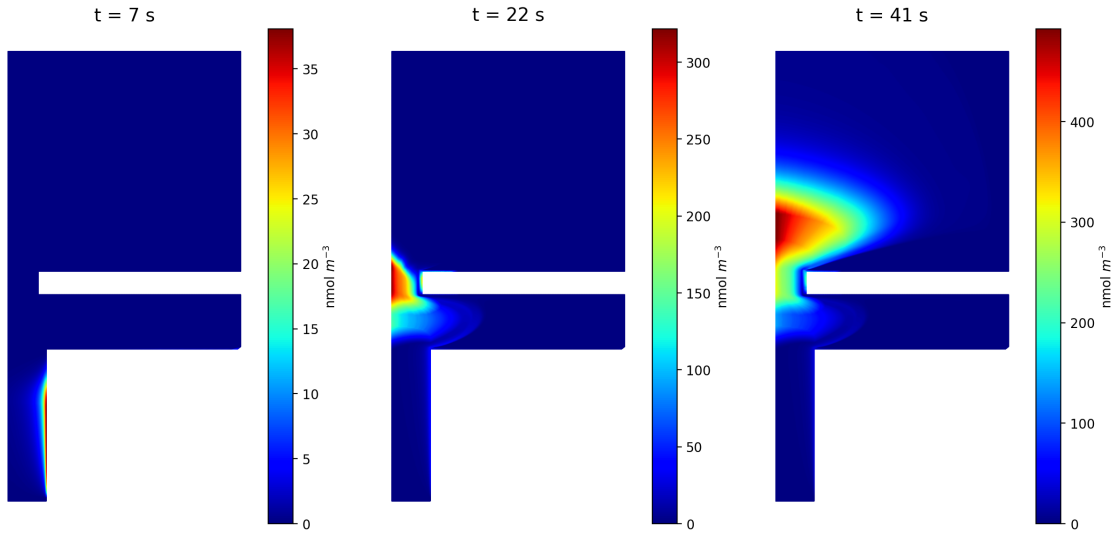

Figure 3: Concentration of vWFs at $t = 7\,s, 22\,s, 41\,s$ of the simulation.

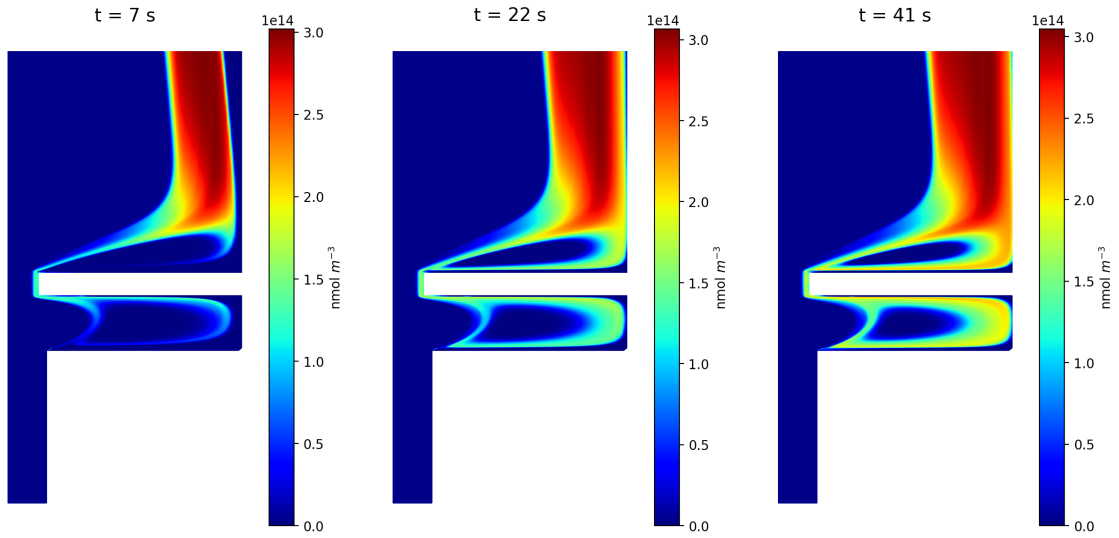

Figure 4: Concentration of AP at seconds $t = 2\,s, 15\,s, 41\,s$ of the simulation.

In Figure 3 and 4 two examples of the vWFs and AP fields at different times, for a given instance of parameters. The dimensions of the medical device can be seen in Figure 1.

2 Deep Learning and Model Order Reduction

Many methods have been developed to avoid solving a **parametrized** system of PDEs by means of a *full order model* (FOM), that is by solving directly the PDEs of the model [9]. Many of these methods fall under the category of *reduced order models* (ROM): the idea is to replace the FOM with a ROM, which features a much lower dimension, but is still able to represent the most important physical features described by the FOM. The ROM is sometimes referred to as the *surrogate model*. Often these methods are based on the assumption that the reduced order approximation is given by a **linear** combination of vectors of a basis, which can be found starting from a set of solutions obtained from the FOM, usually called **snapshots**. One of the most famous and deployed technique is the Proper Orthogonal Decomposition (POD), which finds a linear basis of vectors, which we refer to as *modes*, on which approximate the FOM solutions. In this case the degrees of freedom (depending both on time and parameters) of the POD modes are obtained from the solution of a low-dynamical system, given by a (Petrov-)Galerkin projection onto a linear test subspace. This approach relies on a *training* and on a *testing* stage: once the training has been completed the low-dimensional space is computed, and during the testing stage an approximation of the FOM can be given for every new parameter instance. This method has some drawbacks: if the problem is highly nonlinear a large number of modes is required, leading to computational problems in order to find the coefficients of the basis [10]. Thus two new approaches has been proposed, exploiting Neural Networks (NN), on the wake of what has been done with Convolutional AutoEncoders (CAE) in [11]: DL-ROM [12] and POD-DL-ROM [13]. DL-ROM exploits a CAE coupled with a Deep FeedForward neural network, looking for the PDEs solutions in a **nonlinear** subspace. POD-DL-ROM is built upon the same structure, but takes advantage of an *a priori* model order reduction stage (through POD) to make the training stage less computationally costly. We will show how thanks to those methods it is possible to build a model that, once trained, is able to construct **the fields** (the biochemical species of the thrombosis model) in time and space for a given **parameter instance**.

2.1 Proper Orthogonal Decomposition (POD)

Since POD is crucial to POD-DL-ROM, we briefly present the mathematics on which it is built upon [14]. POD was firstly introduced by Lumley [15], with the goal of decomposing the random vector field of a turbulent fluid motion into a set of deterministic functions, such that each of them highlighted a part of the total turbulent flow. The core idea was to separate the main coherent structures that altogether give raise to the turbulent flow. Let us call $\mathbf{u}'(\mathbf{x}, t)$ the fluctuating velocity of the field depending on time and space, with $\mathbf{u}'$ the velocity vector $\mathbf{U}$ minus its temporal mean. The aim of POD is to find a collection of deterministic spatial functions $\mathbf{\Phi}_k(\mathbf{x})$ (the modes) and random time coefficients $\alpha_k(t)$ such that

$$\mathbf{u}'(\mathbf{x}, t) = \sum_{k=1}^{\infty} \alpha_k(t) \mathbf{\Phi}_k(\mathbf{x}). \tag{4}$$

The way the coefficients are selected is such that the sum $\sum_{k=1}^{N} \alpha_k(t)\mathbf{\Phi}_k(\mathbf{x})$ maximizes the kinetic energy that can be captured by the first N modes. Furthermore the modes are orthogonal, thus constituting an orthogonal basis. A common way to practically construct the orthogonal basis makes use of Singular Value Decomposition (SVD) through the *snapshot* method [16]. We start by considering a collection of *snapshots* $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_s}\}$ for a given field. Every snapshot $\mathbf{x}_j \in \mathbb{R}^{N_h}$ (obtained from a simulation, for example via a finite volume method) is a solution of the system dynamics at different time and/or for different set of parameters. N_h is the size of the mesh on which the simulations are performed. The next step is to build the so called *snapshot matrix* $\mathbf{S}^{N_h \times n_s}$, whose jth column is given by $\mathbf{x}_j$. SVD is then computed on $\mathbf{S}$, so that

$$\mathbf{S} = \mathbf{U}\mathbf{\Sigma}\mathbf{Y}^T \tag{5}$$

where the columns of the matrix $\mathbf{U} \in \mathbb{R}^{N_h \times n_s}$ are the left singular vectors of $\mathbf{S}$ and the columns of $\mathbf{Y} \in \mathbb{R}^{n_s \times n_s}$ are its right singular vectors. $\mathbf{\Sigma} \in \mathbb{R}^{n_s \times n_s} = diag(\sigma_1, \sigma_2 \ldots \sigma_{n_s})$, such that $\sigma_1^2 \geq \sigma_2^2 \geq \ldots \geq \sigma_{n_s}^2 \geq 0$ are called the singular values of $\mathbf{S}$, and each of them represents the amount of energy expressed by its associated left singular vector (σ_1 is associated to the first left singular vector and so on). The first N left singular vectors (the modes) form the POD basis $\mathbf{V}_N$, which is orthogonal. The POD basis is said "optimal", since among all the orthonormal basis of size N, it is the one which minimizes the least squares error of snapshot reconstruction,

$$\min_{\mathbf{V}_N \in \mathbb{R}^{N_h \times N}} ||\mathbf{X} - \mathbf{V}_N\mathbf{V}_N^T\mathbf{X}||_F^2 = \min_{\mathbf{V}_N \in \mathbb{R}^{N_h \times N}} \sum_{i=1}^{n_s} ||\mathbf{x}_i - \mathbf{V}_N\mathbf{V}_N^T\mathbf{x}_i||_2^2 = \sum_{i=N+1}^{n_s} \sigma_i^2. \tag{6}$$

From 6 we can say that the square error is given by the sum of the squares of the singular values corresponding to the left singular vectors not in $\mathbf{V}_N$. Thus in order to choose the first N left singular vectors, N is usually chosen such that

$$\frac{\sum_{i=1}^{N} \sigma_i^2}{\sum_{i=1}^{n_s} \sigma_i^2} > \epsilon \tag{7}$$

with ϵ arbitrary. Refer to [17] for a detailed analysis of POD reconstruction error. Once the POD basis $\mathbf{V}_N$ has been computed, the next step is to find the right coefficient to give to the basis in order to reconstruct $\mathbf{u}'(\mathbf{x}, t)$ as a linear combinations of $\mathbf{V}_N$. As it will be explained better in what follows, in this work the idea is to collect a series of snapshots, which vary both in time and parameters, from the thrombosis simulation. The task of finding the correct coefficients for $\mathbf{V}_N$, given time and a certain set of parameters as input, will be left to Neural Networks.

2.2 Deep Feed-Forward Neural Networks (DFNN), Convolutional Neural Networks (CNN) and Autoencoders (AE)

What follows [1] is a brief introduction to some Machine Learning (ML) structures important for this work, a detailed theoretical explanation is out of the scope of this

[1] The images used in this section are taken from [18].

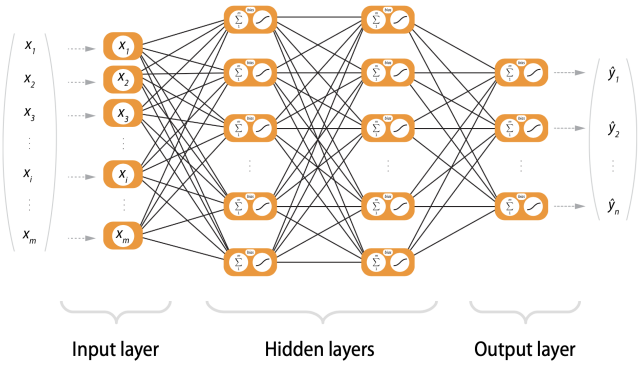

Figure 5: A dense DFNN architecture. Every neuron of every layer takes as input the output of the previous layer and applies to it a linear transformation followed by a non-linear one.

report and can be found in [19], [20], [21] and [22]. The Neural Network (NN) architecture we use is made up of two main components: a DFNN and a Convolutional AE. DFNN are neural-inspired nonlinear models for supervised learning. They are able to mimic the behavior of any non-linear function if enough data are provided. The basic unit of a NN is a 'neuron' i, which receives as input a vector of k features $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ and gives as output a scalar $a_i(\mathbf{x})$. A collection of neurons forms a layer, and a series of layers makes a NN, with the output of a layer serving as the input for the next. The first layer is the *input layer*, the middle layers are the *hidden layers* and the final layer is the *output* layer. The transformation a_i is composed of two parts: an initial linear mapping that weights the importance of each feature, and a non-linear mapping σ_i. The linear mapping is a simple dot product followed by a re-centering with a neuron-specific bias b^i:

$$z^i = \boldsymbol{\omega}^i \cdot \mathbf{x} + b^i. \tag{8}$$

After this the non-linear function is applied, so $a_i(\mathbf{x}) = \sigma_i(z^i)$. If the layers are dense, see Figure 5, each neuron of a layer takes as input the output of the previous layer and applies to it the scalar product of its own weights together with its own non-linear transformation (which is usually the same for all the neurons of a given layer). The power of neural networks lies in the universal approximation theorem which states that a neural network with a single hidden layer can approximate any continuous, multi-input/multi-output function with arbitrary accuracy.

Convolutional Neural Networks were invented primarily for computational reasons in the field of image recognition: as the number of pixels increases, the cost of using dense neural networks becomes prohibitive (i.e. there are too many unknown parameters to determine). As it is said in [19], *Convolutional neural networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.* Convolutional networks have sparse interaction, meaning that the input image may have millions of pixel, but the output may occupy only hundreds of pixels, thanks to the use of filters which detect the real important features of the images (such as edges or particular shapes). The mathematical tool exploited is finite convolution, which is realized through the use of filters, as it is

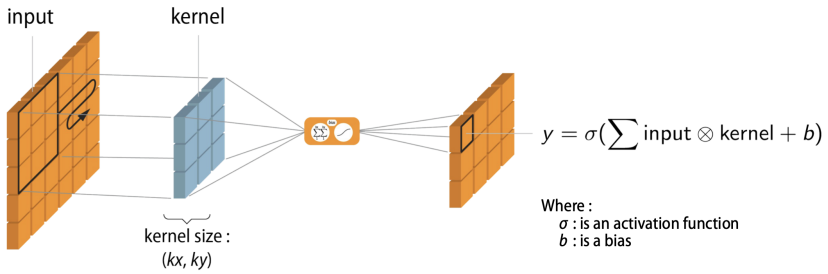

Figure 6: The application of convolution through the use of a kernel (filter), followed by a non linear transformation σ.

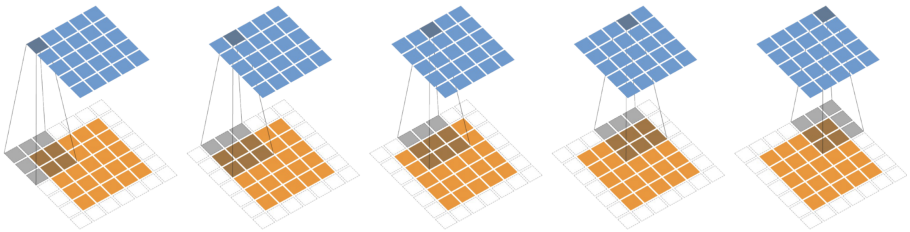

Figure 7: Padding with a stride of 1.

shown in Figure 6. The kernel (filter) is here a 3×3 matrix, as it is customary in CNN architecture, although only by trying is it possible to find which kernel size gives the best accuracy. Letting m be the kernel height and n its width, applying the filter on a section of the Input gives the following scalar: $y' = \sum_{i=1}^{m} \sum_{j=1}^{n} \omega_{i,j} x_{i,j}$. Here $\omega_{i,j}$ is the weight associated to the element i, j of the kernel and $x_{i,j}$ the pixel in position i, j of the Input. Subsequently, a bias b is sum to y' and a non-linear transformation σ is applied to it, so that $y = \sigma(y' + b)$. By making the kernel slide on the Input a final output is obtained, whose dimension is determined by two CNN parameters: the *stride* and the *padding*. The stride defines of how many pixel the kernel moves at each move (a stride of two will reduce by two the output size), while the padding determines if all the pixels are taken into consideration (to make it possible to perform a convolution on the pixels on the border some zeroes are added outside the image as in Figure 7). At every convolutional layer many different filters can be applied, and at the end the goal is to get a final image which contains the most important features of the initial Input.

AutoEncoders are a NN structure built on the idea of reconstructing the input given. This may be useful in case there are some images with noise and we want to re-build them without the noise. In Figure 8 we can see how in the Encoder the image input gets reduced in size more and more until the latent space, where the important features of the original image should be recognized. Subsequently the Decoder reconstructs the original image from the latent space, but only displaying its main features (without the noise). The layers that make the AutoEncoder can be whatever, although (as in this work) they are usually convolutional, because of their computationally efficiency and ability at recognizing patterns. In this case they are

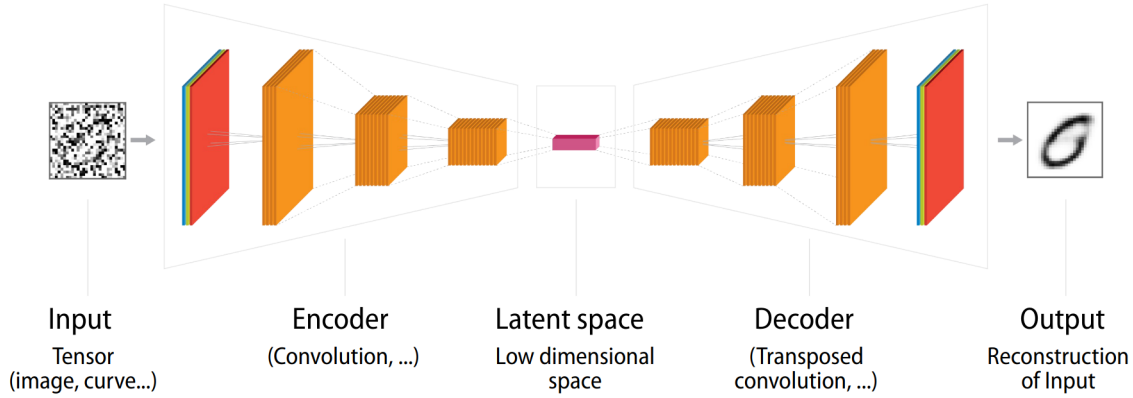

Figure 8: Structure of an AutoEncoder.

called *convolutional AutoEncoders*.

To conclude, no matter the architecture chosen, it will depend on a lot of unknown parameters (present in layers, filters, etc. in form weights, biases, etc.) to be determined. This is done during the *training stage*, thanks to the loss function (which compares the ML output with a known expected result) and algorithms such as Stochastic Gradient Descent and Adaptive Moment Estimation (which change the parameters to push the loss function towards a minimum) [23].

2.3 Problem formulation

Introducing one discretization technique, such as the Finite Element method [4], the FOM can be described as a nonlinear parametrized dynamical system. Given a set of parameters $\boldsymbol{\mu}$ (in our case Table 2), we want to solve the initial value problem:

$$\begin{cases} \dot{\mathbf{u}}_h(t,\boldsymbol{\mu}) = \mathbf{f}(t,\mathbf{u}_h(t;\boldsymbol{\mu});\boldsymbol{\mu}) \quad t \in (0,T), \\ \mathbf{u}_h(0;\boldsymbol{\mu}) = \mathbf{u}_0(\boldsymbol{\mu}) \end{cases} \tag{9}$$

where $\mathbf{u}_h$ is the parametrized solution of (9) (the field), $\mathbf{u}_0$ is the initial condition and $\mathbf{f}$ is a nonlinear function which describes the system dynamics (in our case (3)). We define N_h as the dimension of the FOM, which corresponds to the finite discretization of the space (the number of points that make up the mesh on which the simulations are performed). We want to achieve a numerical approximation of the set

$$S_h = \{\mathbf{u}_h(t,\boldsymbol{\mu}) | t \in [0,T), \boldsymbol{\mu} \in \mathbb{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h}, \tag{10}$$

of solutions of (9) when $(t;\boldsymbol{\mu})$ varies in $[0,T) \times \mathbb{P}$. If we assume that the problem admits a unique solution, then the *intrinsic dimension* of (10) is at most $n_{\boldsymbol{\mu}} + 1 << N_h$, with $n_{\boldsymbol{\mu}}$ being the number of parameters, counting time as an additional parameter. We will refer to the approximation of $\mathbf{u}_h(t,\boldsymbol{\mu})$ as to $\tilde{\mathbf{u}}_h(t,\boldsymbol{\mu})$.

2.4 Deep Learning - Reduced Order Model (DL-ROM)

DL-ROM [12], is a method that aims at finding nonlinearly a ROM starting from a FOM. It is a non-intrusive method, meaning that it does not need to really enter

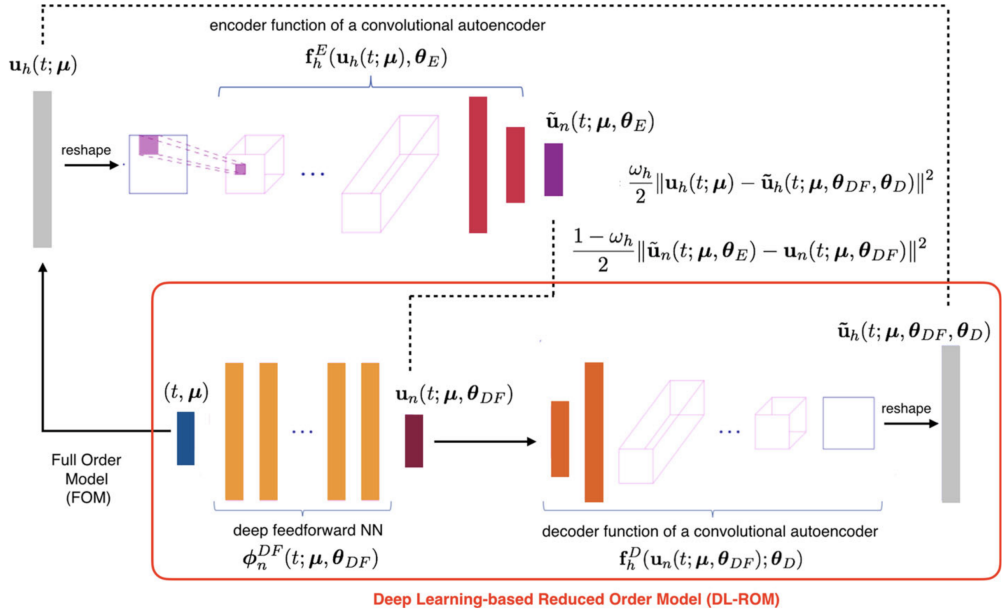

Figure 9: DL-ROM architecture, composed of two interacting branches: a CAE and a DFNN. Notice the loss function is made up of two components. Image from [12].

into the details of the FOM. As a matter of fact, in principle it can be used without even knowing on which equations the simulations are based on: we only need the **snapshots** from the simulations. As stated initially we want to be able to generate the fields in time and space of some biochemical species giving as inputs only the time and the parameters on which the PDEs (CDR equations (3)) depend. DL-ROM is an attempt to solve this problem exploiting a CAE and a DFNN, as it can be seen in Figure 9. This method differs from the POD method, where the approximation of the real solutions of (9) is

$$\mathbf{u}_h(t,\boldsymbol{\mu}) \approx \tilde{\mathbf{u}}_h(t,\boldsymbol{\mu}) = \mathbf{V}_N \mathbf{u}_N(t,\boldsymbol{\mu}), \tag{11}$$

where $\mathbf{V}_N$ is the matrix with the singular vectors of the SVD (5) decomposition, $\mathbf{u}_N$ are the coefficients of the linear basis and N is the dimension of the basis. In POD based approach, as it will be clear in POD-DL-ROM, the most difficult part is finding the coefficients $\mathbf{u}_N$. In DL-ROM instead, the approximation is:

$$\mathbf{u}_h(t,\boldsymbol{\mu}) \approx \tilde{\mathbf{u}}_h(t,\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D) = \mathbf{f}_h^D(\Phi_n^{DF}(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF});\boldsymbol{\theta}_D). \tag{12}$$

In order:

- $\boldsymbol{\phi}_n^{DF}(.;.;\boldsymbol{\theta}_{DF}) : \mathbb{R}^{n_\mu+1} \to \mathbb{R}^n$ is a DFNN which takes as argument a vector $(t;\boldsymbol{\mu})$ of dimension $n_{\boldsymbol{\mu}}+1$ which contains time and parameters (this is exactly the aim of this work: giving as input time and parameters of the PDEs and getting the field in space). The DFNN is responsible for the *reduced dynamics learning*, i.e. it gives the intrinsic coordinates, $\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF})$, of the ROM approximation in a reduced nonlinear manifold (as opposed to the POD which exploits a linear manifold). Notice that the DFNN is used both at *training* and *testing* stage.

- $\mathbf{f}^D(.;\boldsymbol{\theta}_D) : \mathbb{R}^n \rightarrow \mathbb{R}^{N_h}$ is the decoder of the CAE, which depends on an ensemble of weights and biases $\boldsymbol{\theta}_D$. The decoder is responsible for the *reduced trial manifold learning*, i.e. it reconstructs the final field, $\tilde{\mathbf{u}}_h(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)$, from the intrinsic coordinates from the DFNN.

- Finally $\mathbf{f}_n^E(.;\boldsymbol{\theta}_E)$ is the encoder, which depends on parameters $\boldsymbol{\theta}_E$. Its job is to be able to capture the main features, $\tilde{\mathbf{u}}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_E)$, of the input $\mathbf{u}_h(t,\boldsymbol{\mu})$, i.e. a *snapshot* from the FOM (obtained from the simulations). The snapshot is reshaped as shown in [9] as a square image before feeding the encoder, in order to exploit the CNN layers. The encoder does not appear in (12) since it is **discarded at testing time**: once the parameters of the structure represented in Figure 9 have been obtained in the training, at testing stage **only** the vector $(t;\boldsymbol{\mu})$ of the DFNN is given as input. The main role of the encoder is to connect through one piece of the loss function $\tilde{\mathbf{u}}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_E)$ with $\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF})$, i.e. the main features of the input snapshot with the intrinsic coordinates of the dynamical system (which are the output of $(t,\boldsymbol{\mu})$ through the DFNN). The main features are enclosed in the *latent vector* of the encoder, whose dimension is close to $n_{\boldsymbol{\mu}} + 1 << N_h$.

In practice during the training **two inputs** are given at the same time: the snapshot obtained by the FOM $\mathbf{u}_h(t,\boldsymbol{\mu})$ and its associated vector $(t;\boldsymbol{\mu})$. The former is reshaped into a square image and goes through some convolutional and dense layers until $\tilde{\mathbf{u}}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_E)$ while the latter goes through some dense layer until $\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF})$. The first part of the loss function (*internal* error) is then computed, $\mathbb{L}_{int}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) = ||\tilde{\mathbf{u}}_n(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_E) - \mathbf{u}_n(t^k;\boldsymbol{\mu},\boldsymbol{\theta}_{DF})||^2$. Subsequently the output of the DFNN $\mathbf{u}_n(t;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF})$ is given as input to the Decoder, until $\tilde{\mathbf{u}}_h(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)$. Then the second part of the loss function (*reconstruction* error) can be computed: $\mathbb{L}_{rec}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) = ||\mathbf{u}_h(t^k;\boldsymbol{\mu}_i) - \tilde{\mathbf{u}}_h(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)||^2$. At the end of this iteration we get the total loss function as:

$$\mathbb{L}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) = \frac{\omega_h}{2}\mathbb{L}_{int}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) + \left(1 - \frac{\omega_h}{2}\right)\mathbb{L}_{rec}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}). \tag{13}$$

Since this process is done for all the snapshots of the training (for one *epoch* in ML jargon), what we want to minimize in order to find the best parameters for the CAE and for the DFNN is:

$$\mathbb{J}(\boldsymbol{\theta}) = \frac{1}{N_s}\sum_{i=1}^{N_{train}}\sum_{k=1}^{N_t}\mathbb{L}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}), \tag{14}$$

where N_s is $N_{train} \times N_t$.

The DL-ROM method is tested on several test-cases by the authors in [12], but the 2D examples present, as they remark, deal with a mesh of a dimension of the order of maximum 10^4, and it still takes a lot of time in the *training* stage. Computational inefficiency in the *training* is the reason why POD-DL-ROM [13] was later proposed. In our case the mesh has 68650 points, and when we attempted to use DL-ROM on our data from thrombosis simulations, it took almost 2 hours to do 140 epochs (with 4000 data-points) and the loss function was still of the order of 10^2, Figure 10.

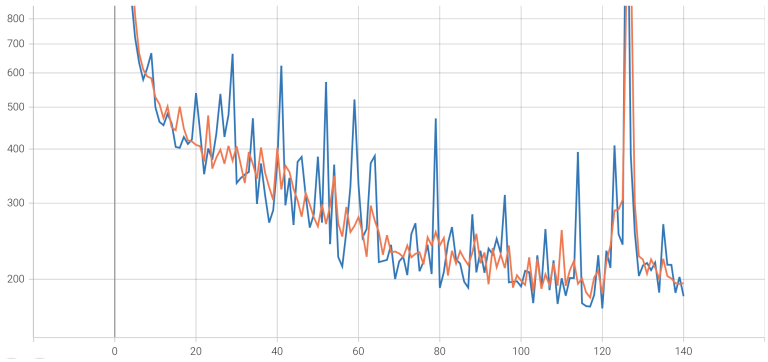

Figure 10: Loss function of DL-ROM algorithm applied to thrombosis data. In blue the validation loss and in orange the training loss as a function of the epoch.

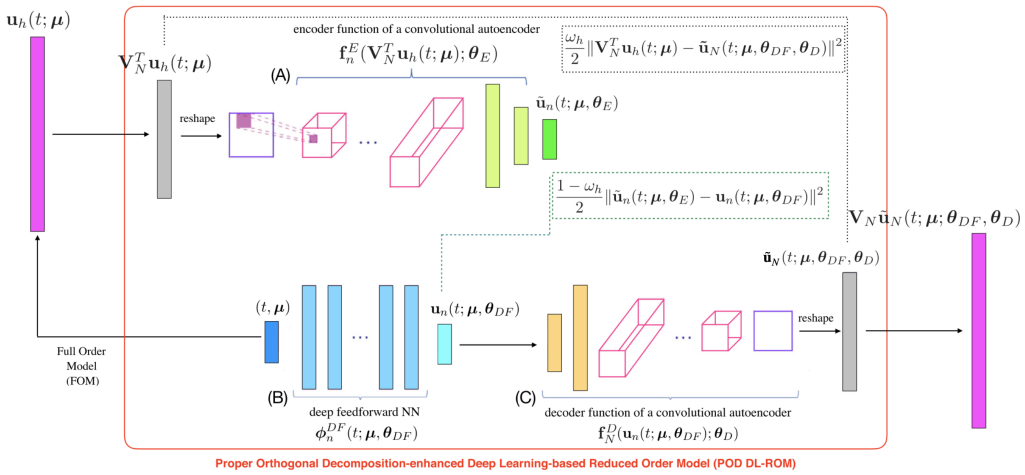

Figure 11: POD-DL-ROM architecture. The difference with DL-ROM (9) is in the input given. Image from [13].

2.5 POD-DL-ROM

POD-DL-ROM [13] has the same identical purpose of DL-ROM, but relies on a **prior dimensionality reduction stage** that aims at improving the DL-ROM *training* stage. The prior dimensionality reduction is performed through POD. As the authors say in [13] at pag. 16, comparing DL-ROM and POD-DL-ROM performances for a given test case, the *training* stage required 23.5 hours for the former and 2.5 hours for the latter, given a mesh of 10657 (we have 68650 points). As it can be observed in Figure 11, the architecture is exactly the one of DL-ROM, but the encoder input now is different: before the encoder inputs reshaped in images were the snapshots from the simulations, while now they are the *coefficients* of the linear basis of the POD on which every snapshot is projected. In other words, here the approximation is, as in the POD approach,

$$\mathbf{u}_h \approx \tilde{\mathbf{u}}_h(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D) = \mathbf{V}_N\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D) \tag{15}$$

and we want to find the coefficients $\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)$. So we give as input to the encoder the $\mathbf{V}^T\mathbf{u}_h$. The procedure is the following:

- A *snapshot matrix* $\mathbf{S}^{N_h,N_s}$ and a *parameter matrix* $\mathbf{M}^{(n_\mu+1),N_s}$ are created, such that the jth column of $\mathbf{S}$ is a snapshot from a simulation with parameters $(t,\boldsymbol{\mu})$ and the column jth of $\mathbf{M}$ contains the same parameters $(t,\boldsymbol{\mu})$. N_h is the mesh size of the simulations and N_s is the number of snapshots from the simulations.

- A SVD (5) is computed on $\mathbf{S}$ as explained in the POD section and the matrix $\mathbf{V}_N$ is obtained, with the first N singular vectors of the SVD as columns.

- Every snapshot of $\mathbf{S}^{N_h,N_s}$ is projected onto $\mathbf{V}_N$, i.e. we do the mapping $\mathbf{u}_h(t;\boldsymbol{\mu}) \rightarrow \mathbf{V}_N^T\mathbf{u}_h(t;\boldsymbol{\mu})$.

- All the process that comes after is the same as described for DL-ROM. Of course the only formal difference is that the *external* error of the loss function is now: $\mathbb{L}_{rec}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) = ||\mathbf{V}_N^T\mathbf{u}_h(t^k;\boldsymbol{\mu}_i) - \tilde{\mathbf{u}}_N(t;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)||^2$, where $\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)$ are the predicted coefficients for the basis (we are now comparing not the fields but the coefficients that will give the fields once combained with $\mathbf{V}$). In fact at the end the prediction of the field will be given by $\mathbf{V}_N\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)$.

The crucial aspect here is that the dimension N of $\mathbf{V}$ is chosen by us: the input dimension of the encoder is equal to the dimension of the POD basis. This is the main improvement: we go from an input dimension with DL-ROM of the order of 10^5 to an order of 10^2, as it will be shown in the 'Results' section. This implies a further subtle aspect: we are using CNN layers to recognize patterns in images which are no more 'real images' (i.e. images of the fields in space as in DL-ROM), but they are a visual representation of the coefficient of the POD basis. In other words, what the ML architecture must learn, is to give the right weights $\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)$ to the POD modes (singular vectors) given a certain $(t,\boldsymbol{\mu})$. Although there is a clear gain in computational complexity, it is natural to wonder if the fact of using a prior dimensionality reduction method can affect the final result. The effect of POD results in loss of information that cannot be retrieved by DL-ROM alone. In a sense, DL-ROM cannot see what has been lost by POD. This aspect is very central and will be discussed in Section (3).

We conclude the theory part with a specification on the way POD is performed: when the snapshot matrix $\mathbf{S}$ becomes large traditional SVD becomes too computationally costly, and thus alternative POD methods have been proposed, such as randomized POD [24], which is used in this work.

2.5.1 Architecture

The results that follow are obtained from the hyperparameters of Tables 3, 4, as in [12]. In Tables 3, 4 the input and output dimensions depend on the size of the layer 1 (N), but the Kernel size, Filters and Stride do not change. The DFNN is composed of 4 dense layers each of them with 100 neurons. We set $\omega_h = 1$ in (13) as done in [12].

Layer	Input Dimension	Output Dimension	Kernel Size	Filters	Stride	Type
1	$[16, 16, 1]$	$[16, 16, 8]$	$[7, 7]$	8	1	CL
2	$[16, 16, 8]$	$[8, 8, 16]$	$[7, 7]$	16	2	CL
3	$[8, 8, 16]$	$[4, 4, 32]$	$[7, 7]$	32	2	CL
4	$[4, 4, 32]$	$[2, 2, 64]$	$[7, 7]$	64	2	CL
5	N	256				Dense
6	256	n				Dense

Table 3: Characteristics of the encoder layers, where $n = n_{\boldsymbol{\mu}} + 1$ and N is the size of the POD basis. 'CL' stands for 'Convolution Layer'

Layer	Input Dimension	Output Dimension	Kernel Size	Filters	Stride	Type
1	n	256				Dense
2	256	N				Dense
3	$[2, 2, 64]$	$[4, 4, 32]$	$[7, 7]$	64	2	(T)CL
4	$[4, 4, 32]$	$[8, 8, 16]$	$[7, 7]$	32	2	(T)CL
5	$[8, 8, 16]$	$[16, 16, 8]$	$[7, 7]$	16	2	(T)CL
6	$[16, 16, 8]$	$[16, 16, 1]$	$[7, 7]$	8	1	(T)CL

Table 4: Characteristics of the decoder layer, where $n = n_{\boldsymbol{\mu}} + 1$ and N is the size of the POD basis. '(T)CL' stands for 'Transposed Convolution Layer'

3 Results

In what follows we show the results of the application of the POD-DL-ROM method to the thrombosis simulation described in Section (1). As already said we want to **predict** in time and space the fields of some biochemical species, giving **as input** $(t, \boldsymbol{\mu})$, i.e. time and PDEs parameters. We will focus mainly on one biochemical specie, vWFs, since it is relevant for thrombus formation and its prediction poses some challenges as we will show. The data we use for the training and testing come from 100 simulations in which a snapshot is saved every second, from $t = 2\,s$ to $t = 41\,s$, for a total of 40 seconds per simulation. For every simulation a set $\boldsymbol{\mu}$ of parameters is sampled uniformly, accordingly to the intervals in Table (2). We will firstly discuss about how data are prepared and normalized, and we will look at the POD basis and coefficients, which are the core of the method. We will then present the results both for fixed parameters and varying time and for fixed time and varying parameters. The problem of early time predictions for weak flows will be then addressed and a possible solution proposed.

3.1 Training and Data preparation

For every result shown a fraction 0.8 of the dataset (100 simulations of 40 snapshot each) is used for training and a 0.2 for the testing. A fraction 0.2 of the training data is used as validation set. The training set is divided in batches of size 40 and a maximum of 10000 epochs (i.e. cycles on the all training set) are performed. An early stop criterion is imposed, i.e. the training is stopped if the loss function of the validation set has not decreased for 500 epochs. We use the ADAM algorithm as an optimizer [25].

Firstly, for a given biochemical species, the snapshot matrix $\mathbf{S}^{N_h,N_s}$ is assembled, where N_h is the mesh size (68650 elements) and N_s is the number of simulations $\times$ the length (in seconds) of every simulation (100×40), such that:

$$\mathbf{S} = [\mathbf{u}_h(t^1;\boldsymbol{\mu}_1)|...|\mathbf{u}_h(t^{N_t};\boldsymbol{\mu}_1)|...|\mathbf{u}_h(t^1;\boldsymbol{\mu}_{N_{train}})|...|\mathbf{u}_h(t^{N_t};\boldsymbol{\mu}_{N_{train}})]. \tag{16}$$

So the first 40 columns are the 40 snapshots of a simulation with a certain set of parameters $\boldsymbol{\mu}_1$, the columns from 41 to 80 are the 40 snapshots of a simulation with a certain set of parameters $\boldsymbol{\mu}_2$ and so on. Then the parameter matrix $\mathbf{M}^{(n_{\boldsymbol{\mu}}+1),N_s}$ is assembled, such that:

$$\mathbf{M} = [(t^1,\boldsymbol{\mu}_1))|...|(t^{N_t},\boldsymbol{\mu}_1))|...|(t^1,\boldsymbol{\mu}_{N_{train}})|...|(t^{N_t},\boldsymbol{\mu}_{N_{train}})]. \tag{17}$$

Notice that the length of every column of $\mathbf{M}$ is $(n_{\boldsymbol{\mu}} + 1)$, i.e. we are simply treating time as an additional parameter. In this way one can find in the jth column of $\mathbf{M}$ the parameters (including time) that generate the snapshot presented in the jth column of $\mathbf{S}$. Subsequently we divide $\mathbf{S}$ and $\mathbf{M}$ in $[\mathbf{S}^{train}, \mathbf{S}^{test}]$ and $[\mathbf{M}^{train}, \mathbf{M}^{test}]$, with a splitting $[0.8, 0.2]$, i.e. 3800 data-points will be used for the *training* and 200 for the *testing* (we recall that only $\mathbf{M}^{test}$ will be used as input during the testing stage. In fact $\mathbf{S}^{test}$ will only be used to compare the predictions made). In this way we can observe what happens when we try to predict from a set of parameters $\tilde{\boldsymbol{\mu}}$ that has never been seen in the *training*. We can also test on some snapshots taken

from a t different from the ones given in the training (if it stays in the interval of time used in the training.)

We then perform the *random* SVD on $\mathbf{S}^{train}$ in order to get the matrix $\mathbf{V}_N$ of singular vectors, and we project every snapshot of $\mathbf{S}^{train}$ onto the basis $\mathbf{V}_N$. In this way we obtain the final Input matrix of coefficients $\mathbf{I}^{N,N_s}$ which will be given as input to the encoder. In doing so we passed from a (*snapshot*) matrix of vectors of dimension N_h to a (*coefficient*) matrix of vectors of dimension N.

Given $\mathbf{I}^{N,N_s}$ and $\mathbf{M}^{train}$, which are respectively the inputs of the encoder and of the DFNN, we normalize them, in order to bring every element in the range [0 1]:

- As far as $\mathbf{I}$ is concerned, we define $\mathbf{I}_{max} = \max_{i=1...N_s} \max_{j=1...N} \mathbf{I}_{i,j}$ and $\mathbf{I}_{min} = \min_{i=1...N_s} \min_{j=1...N} \mathbf{I}_{i,j}$. We then rescale every element of $\mathbf{I}$ according to the following mapping:

$$\mathbf{I}_{i,j} \rightarrow \frac{\mathbf{I}_{i,j} - \mathbf{I}_{min}}{\mathbf{I}_{max} - \mathbf{I}_{min}}. \tag{18}$$

  In practice, we look for the *max* and the *min* of $\mathbf{I}$ across all the elements of every column and we rescale everything given them as it is done in [11].

- As far as $\mathbf{M}^{train}$ is concerned, we define $M^k_{max} = \max_{i=1...N_s} \mathbf{M}^{train}_{k,i}$ and $M^k_{min} = \min_{i=1...N_s} \mathbf{M}^{train}_{k,i}$, i.e. we obtain the *max* and the *min* of every row. We then apply the following mapping:

$$\mathbf{M}^{train}_{i,j} \rightarrow \frac{\mathbf{M}^{train}_{i,j} - \mathbf{M}^{i}_{min}}{\mathbf{M}^{i}_{max} - \mathbf{M}^{i}_{min}}. \tag{19}$$

  In words, we normalize every element according to the *max* and the *min* of the feature class to which it belongs (we no more look for the global *max* and global *min* as for $\mathbf{I}$). This is important because different class of parameters (every feature) have different ranges, sometimes varying of orders of magnitude, as it can be seen in Table (2).

During the testing stage we will use the *same* statistics, i.e. the same *max* and *min* values found, to normalize $\mathbf{M}^{test}$ and to 'inverse normalize' (i.e. applying the inverse formula of (18)) the predicted coefficients $\tilde{\mathbf{u}}_N(t; \boldsymbol{\mu}_i, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$. We remark that it is fundamental to use at testing time the same statistics used to normalize the elements for the training.

Before finally using the columns of $\mathbf{I}$ as inputs, they must be reshaped into images as showed in Figure 11. The way it is done is the following: a vector $\mathbf{x}$ of dimension d is reshaped into a square image $\mathbf{X}^{\sqrt{d},\sqrt{d}}$. In practice the first $\sqrt{d}$ elements become the first row of $\mathbf{X}$, the second $\sqrt{d}$ elements the second row and so on. If $d \neq 4^m$ with $m \in \mathbb{N}$, the input is zero-padded [19]. Notice that in this way the input of the 5th layer of the encoder and the output of the 2nd layer of the decoder have dimension exactly N (Table 3,4).

3.2 Initial analysis of Data

To start with, let us understand by visualization what are the inputs we are using. In Figure 12 we can see the normalized images that are given as input to the encoder,

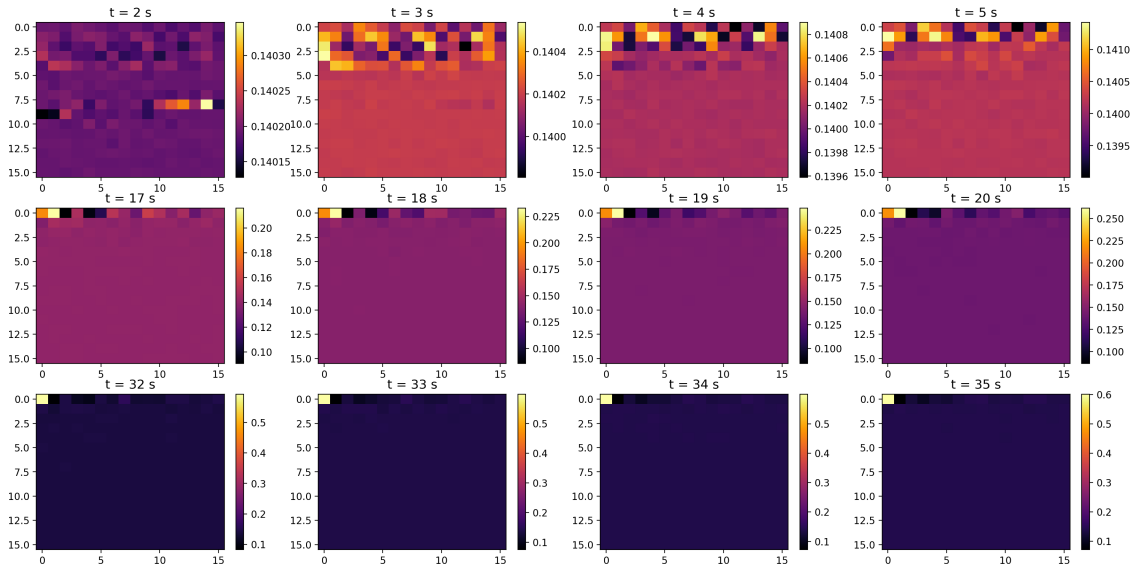

Figure 12: Normalized image input of the encoder (training stage) for vWFs, for a fixed set of parameters at varying time. Notice that a large number of POD modes are necessary at the initial time steps, whereas only the first POD mode is activated at the end of the simulation ($t > 30\,s$).

for vWFs. In this case $N = 256$ (the size of the POD basis $\mathbf{V}_N$), and so the images are of size $\sqrt{256} \times \sqrt{256}$. We recall that these images are a visual representation of the coefficients obtained by projecting a given snapshot onto $\mathbf{V}_N$. In a sense, these images give an idea of which modes are the most important for a certain snapshot at time t and parameters $\boldsymbol{\mu}$. As we can observe in Figure 12, $t = 2\,s$ has as biggest coefficient the one in the 142nd position: this means that the 142nd mode is the most similar to the snapshot of $t = 2$, among all the ones in $\mathbf{V}_N$. From $t = 3\,s$ to $t = 5\,s$ instead the important modes become sparser, while towards the end of the simulation the first mode is the only one that really matters. Since those images are the ones that go through the convolutional layers, it appears logical that the last seconds of the simulations will be more easily predicted, since there is only one clear coefficient to be determined. (However this is not a problem of *classification*: the algorithm does not have to recognize 'the most important mode', but it has to give it the right weight, and this weight will vary, for fixed time, at the variation of $\boldsymbol{\mu}$).

A natural question follows: in Figure 12 we see only the input images for a fixed $\boldsymbol{\mu}$, but in which way do those images change when $\boldsymbol{\mu}$ is varied? In other words, will the shape of vWFs change for fixed time at the variation of $\boldsymbol{\mu}$, or there will be only a variation in its *concentration*? In the first case we would have also a variation in the 'most important' modes, while in the second case they will be the same but with a different assigned weight. Figure 13 shows the normalized standard deviation across the 100 simulation for $t = 12\,s$ and $t = 41\,s$ and the corresponding snapshot from one random simulation: we can see how the variation indeed appears mainly in the concentration, not in the shape. Thus what we expect is that the inputs of Figure 12 for different simulations will select the same 'important' coefficients but with different weights. This makes sense physically: what we are simulating, up to $t = 41\,s$ is before the thrombus formation. Thus the blood flow is the same for every

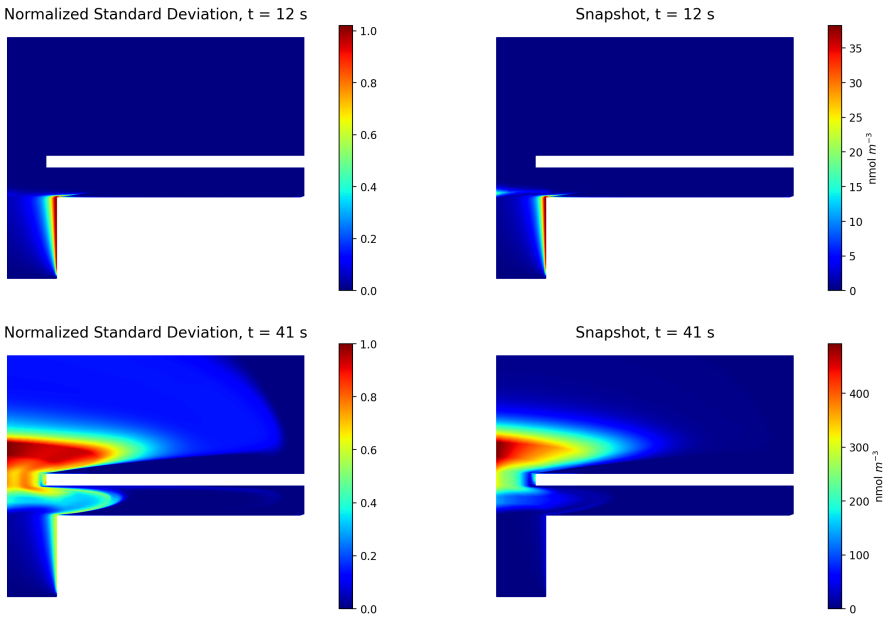

Figure 13: Standard deviation normalized of vWFs across the 100 simulations, compared with random snapshot at the same t. The normalization is done similarly to (18), whit the max and the min taken over the values of the standard deviation performed at fixed time across simulations.

simulation, bringing to a similar shape for a chosen biochemical specie across the simulation. However what we expect is that a different value in concentration will lead to a different shape of the thrombus, which ultimately will result in a different shape of blood flow and thus of the biochemical specie.

We have looked at the coefficients and to the snapshot of the FOM, now let us have a better intuition about the singular vectors which are the columns of $\mathbf{V}_N$. As explained in Section (2.1), at every singular vector j we can associate a singular value σ_j, which quantifies how much 'energy' is associated to the jth mode, i.e. how much information it gives about the global flow. Notice that if we performed the SVD on a snapshot matrix at varying time but at fixed $\boldsymbol{\mu}$, the modes would show the main flows in time. Here we have a snapshot matrix (16) at varying **time** and **parameters**, so also the main variations in parameters are encoded in the modes. In Figure 14 the normalized singular values of AP and vWFs are plotted. We can see that there are 5 singular values $> 10^3$ are for AP, while 10 for vWFs. This means that we can expect a greater variability in vWfs (although we cannot say a priori if the greater variability is in time or in parameters). To conclude, we plot in Figure 15 the first, the second and the 142th mode of vWFs, which according to Figure 12 should be close in shape to the snapshots respectively at the end, at the middle and at the beginning of a fixed simulation (remember that this reasoning makes sense since we have variability across simulations in concentration but not much in shape). Figure 15 suggests that $t = 41\,s$ will be easier to predict than the previous snapshots, since there is a mode (the first), which is really close in shape to it. Be aware that this analysis is deeply system dependent: varying the biochemical species analysed will surely result in a different Figure 12.

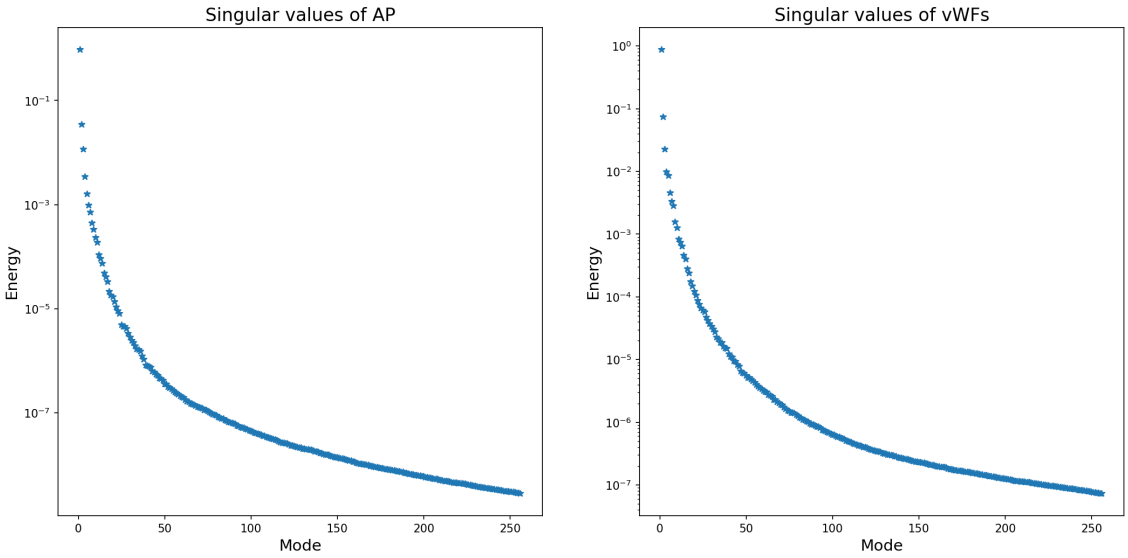

Figure 14: Normalized singular values of AP and vWFS. AP has 5 singular values associated with energy $> 10^{-3}$, vWFs has 10.

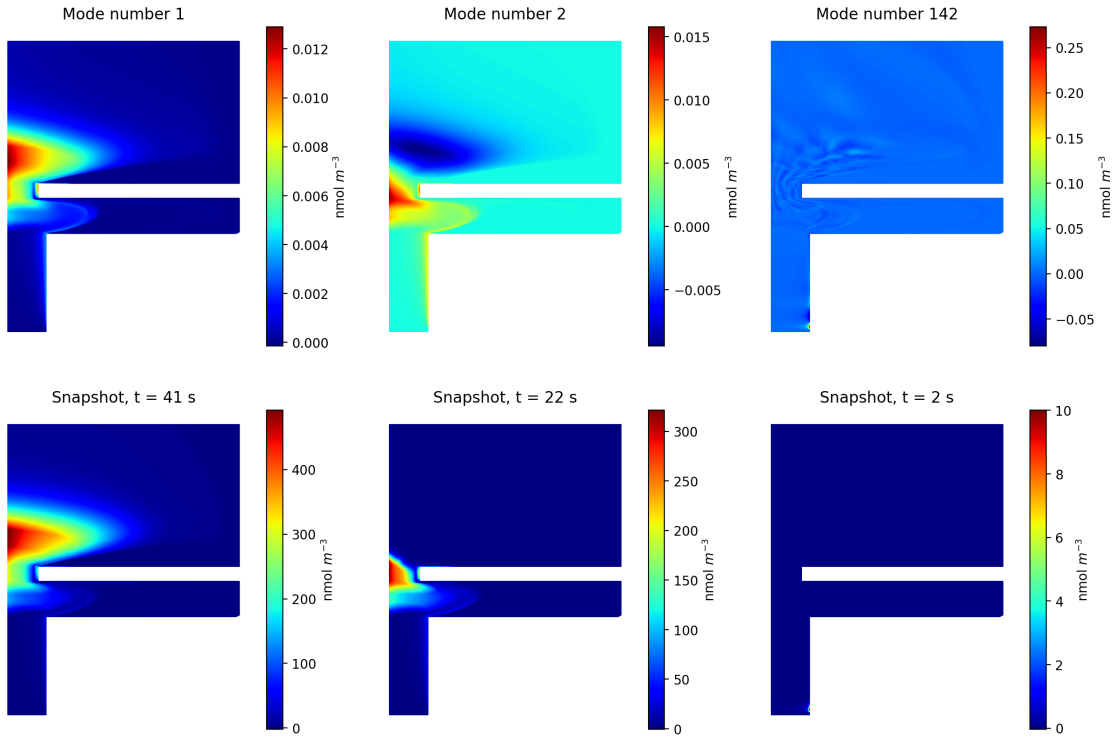

Figure 15: Modes 1, 2 and 142 compared to snapshots of a random simulation of $t = 41\,s, 22\,s$ and $2\,s$.

3.3 Predictions in time and parameters

Here we show the predictions made using the POD-DL-ROM algorithm. The data are taken from 100 simulations and the snapshots are saved every second from $t = 2\,s$ to $t = 41\,s$, before the thrombus formation. The parameters varied randomly (uniformly) in every simulation are the ones in Table (2). In Figure 16 we show the POD-DL-ROM predictions at different times next to the FOM **testing** snapshots, with $N = 64$, where N is the size of the POD basis. We recall that the predictions are the output of the input $(t, \boldsymbol{\mu})$, and the FOM **testing** snapshot (obtained by the simulations) is used to compare the prediction and does not play any role in the POD-DL-ROM prediction (since we are at **testing** time). The relative error displayed is a vector quantity specific for a given snapshot of a given simulation:

$$\boldsymbol{\epsilon}_k = \frac{|\mathbf{u}_h^k(\boldsymbol{\mu}_{test}) - \tilde{\boldsymbol{u}}_h^k(\boldsymbol{\mu}_{test})|}{\sqrt{\frac{1}{N_t}\sum_{k=1}^{N_t}\|\mathbf{u}_h^k(\boldsymbol{\mu}_{test})\|^2}}. \tag{20}$$

We will use also another metric, the global error, which is a scalar value used to asses globally the validity of the predictions in time and parameters:

$$\epsilon_G(\mathbf{u}_h, \tilde{\mathbf{u}}_h) = \frac{1}{N_{test}}\sum_{i=1}^{N_{test}} \frac{\sqrt{\sum_{k=1}^{N_t}\|\mathbf{u}_h^k(\boldsymbol{\mu}_{test,i}) - \tilde{\boldsymbol{u}}_h^k(\boldsymbol{\mu}_{test,i})\|^2}}{\sqrt{\sum_{k=1}^{N_t}\|\mathbf{u}_h^k(\boldsymbol{\mu}_{test,i})\|^2}}, \tag{21}$$

where k is the time chosen, $\boldsymbol{\mu}_{test}$ refers to a specific set $\boldsymbol{\mu}$ of parameters used for testing, N_t is the number of seconds per simulation and N_{test} is the number of different simulations kept for the testing stage. We recall that $\mathbf{u}_h$ is the testing snapshot from the FOM (obtained via a simulation), and $\tilde{\mathbf{u}}_h$ is its POD-DL-ROM prediction. We can also define the relative error **per second**:

$$\epsilon_{sec}(\mathbf{u}_h^t, \tilde{\mathbf{u}}_h^t) = \frac{1}{N_{test}}\sum_{i=1}^{N_{test}} \frac{\|\mathbf{u}_h^t(\boldsymbol{\mu}_{test,i}) - \tilde{\boldsymbol{u}}_h^t(\boldsymbol{\mu}_{test,i})\|}{\|\mathbf{u}_h^t(\boldsymbol{\mu}_{test,i})\|}, \tag{22}$$

which is equal to (21) but with only one k, the one of the second t considered (so in the internal sum we sum only on one element). In Figure 17 we see the relative error in time per second for $N = 64, 256, 1024$. The global error in the three cases varies as $\epsilon_G = 0.0323, 0.0364, 0.3663$. So we have that overall the algorithm performs better in the case $N = 64$, although Figure 17 shows that the case $N = 256$ gives better results for the early times and slightly worse for the last seconds. $N = 1024$ instead gives worse results both overall and at every second. This is surprising, as we would expect that having a bigger basis would lead to a better performance. However we should not forget that we are training a ML algorithm, whose accuracy may vary if the Input data given in the training are different. Let us look at the lowest value of the loss function during the training: respectively for $N = 64, 256, 1024$ it is: $3.812^{-5}, 4.24^{-5}, 1.0556^{-4}$. So, although the size of $\mathbf{V}_N$ is increasing, the accuracy of the ML algorithm is decreasing, probably because since the input image is becoming bigger more parameters have to be determined in the training and thus more data are required. But then why does $N = 256$ performs better than $N = 64$ at early

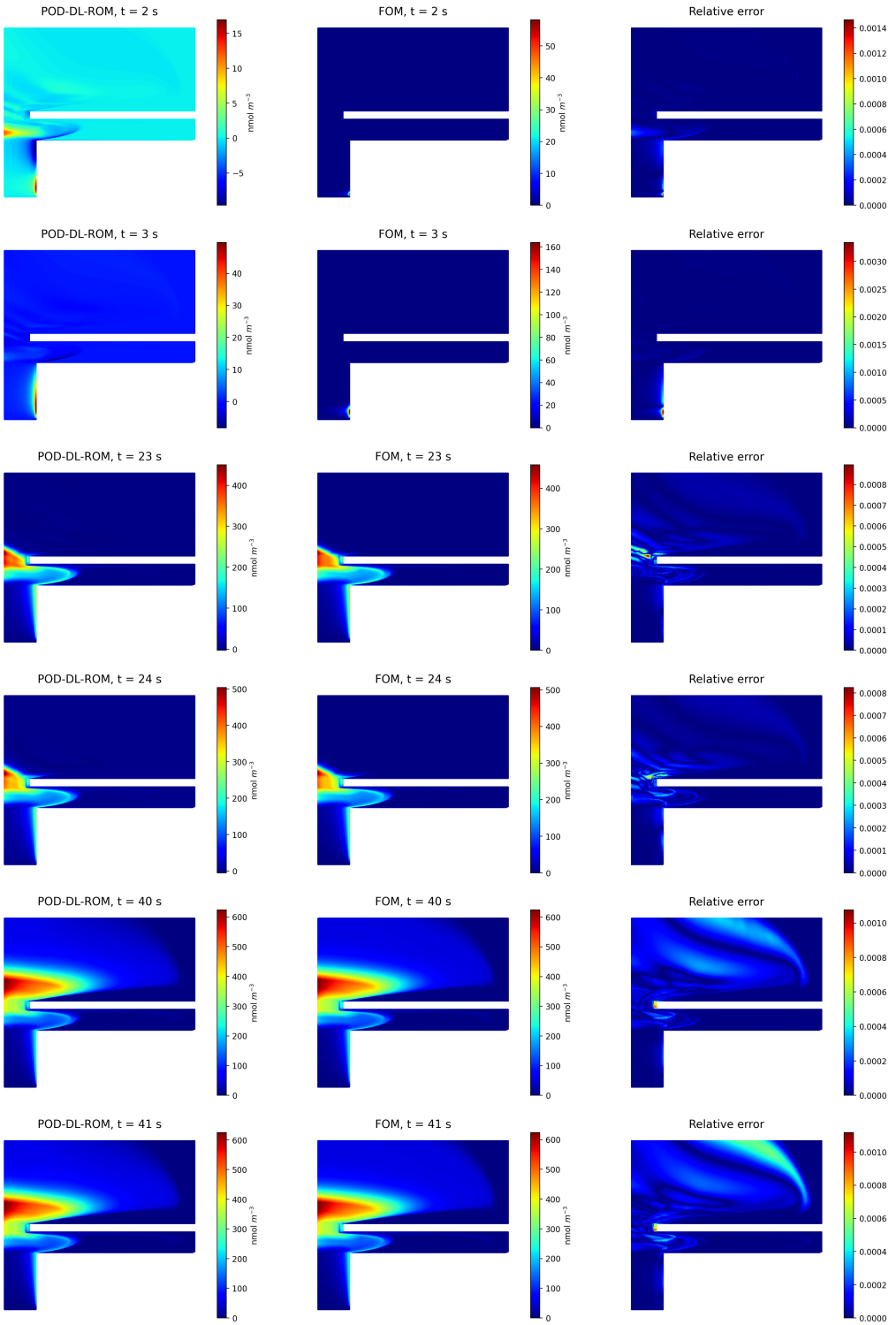

Figure 16: Comparison of vWFs snapshots for a fixed testing $\boldsymbol{\mu}$ between POD-DL-ROM prediction and FOM, at initial, middle and final seconds. The size of the POD basis is $N = 64$.

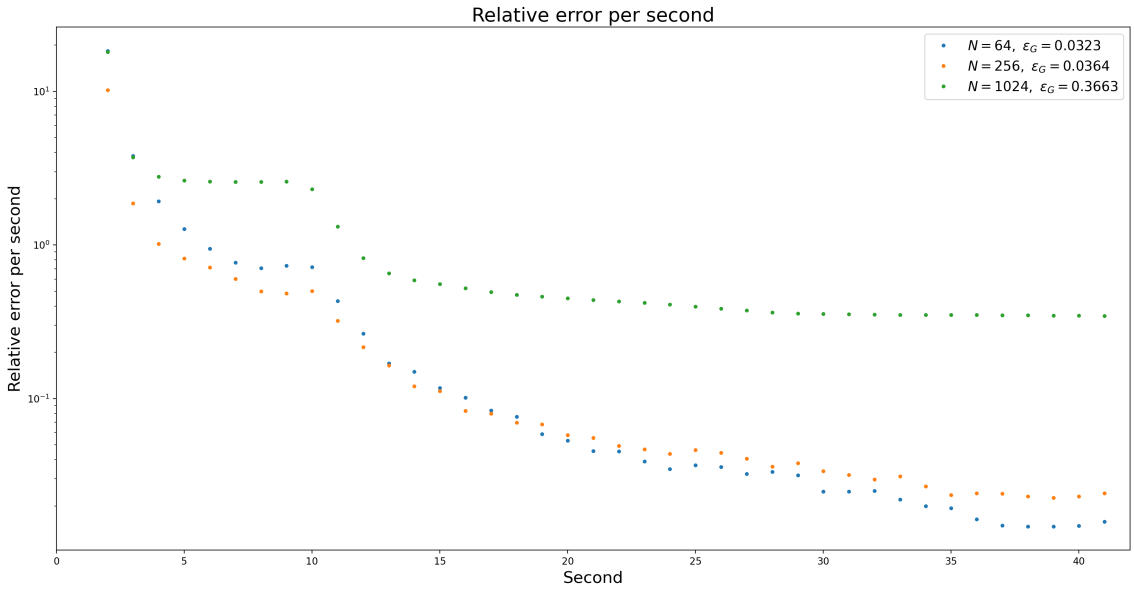

Figure 17: Relative error per second (22) for $N = 64, 256, 1024$, and global error ϵ_G (21) (in the legend).

times but worse at late times? Looking at Figure 12 we can see how the first seconds, especially $t = 2$, need modes that are beyond $N = 64$, and that we have only with $N = 256$. So having more modes is more beneficial than having a greater ML accuracy. The last seconds instead only need the first mode, and so they are predicted with almost equal precision by the three test cases. From these results it looks like there are two main different regimes captured by the POD modes: the initial concentration which rapidly stretches towards the membrane of the device and the final stationary shape well captured by the first mode. Since there are a only a few seconds (the early times) in the training set which need the lonely 142nd mode or a big combination of initial modes, the ML algorithm sees them as outliers and struggles to give them the correct coefficients. Furthermore, if we look closely at the real values of the coefficients in Figure 12, we can see how, for a given snapshot, the coefficient values change from one to the other at the fourth decimal point! At the contrary, the last seconds coefficient vary at the fourth decimal point. This is due to the normalization (18), which normalizes every element of the input matrix $\mathbf{I}$ by the global maximum and global minimum: in our case the early time coefficients are much smaller than the late times coefficients, and thus they become even smaller after the normalization. For this reason it is difficult for the CNN layers to extract correctly features from the early times images, as opposed to the late ones. We now show the predictions for a fixed time ($t = 41\,s$) and different parameters from different testing simulations in Figure 18. The predictions are good both in the concentration and in the shape. In fact the different values of concentrations are well respected, but also the little differences in shape, as the little stripe in the bottom-right of the first simulation which is absent in the last.

To understand whether the bad initial predictions are due to a biochemical specie which varies too abruptly in time, we can look at one, vWFc, which evolves more gradually in time. In Figure 19 we see its image coefficient inputs for a given

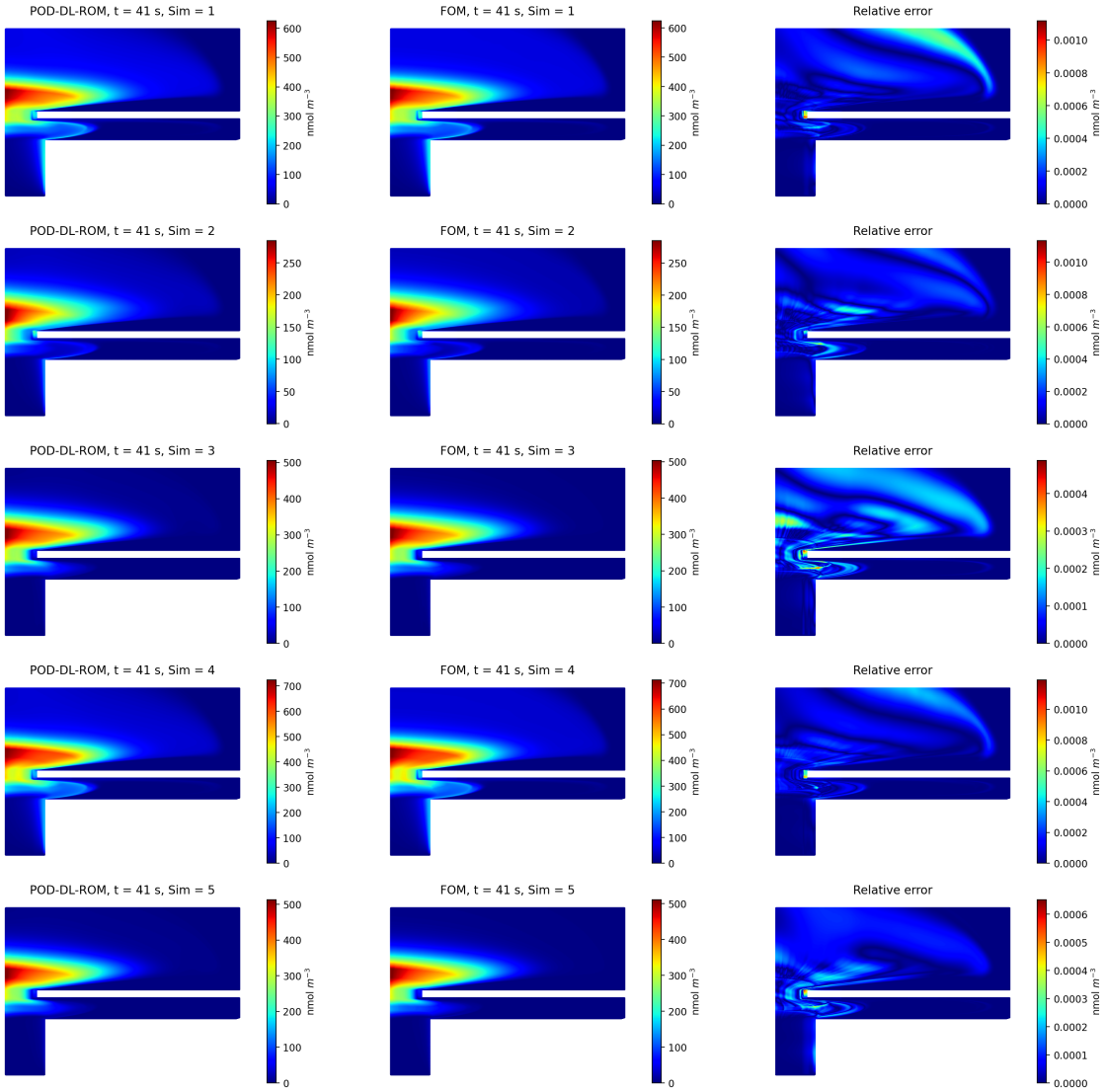

Figure 18: vWFs predictions for fixed time $t = 41\,s$ and varying parameters of Table 2.

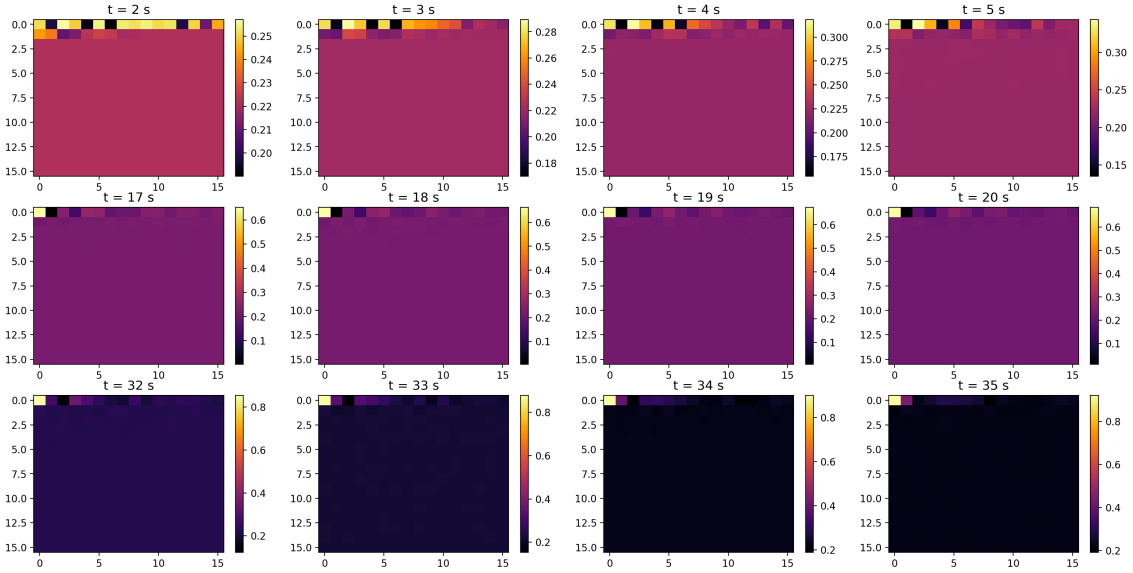

Figure 19: Normalized images inputs (of the encoder) for vWFc, for a fixed simulation (fixed $\boldsymbol{\mu}$) at varying time.

simulation. In this case we do not have anymore $t = 2$ with a very far principal mode, but rather a mixture of only the first 16 modes. In addition, we can see how, for a given snapshot, the different coefficients now vary also for the early times at the first/second decimal point. In Figure 20 we can see how better the predictions are in in this case also at early times. The idea that we need few and well defined coefficients as input images to get good results is what justifies the idea of next section.

3.4 Localized reduced order models

In this section we will change slightly the way we perform POD before the training, in order to make the early time inputs of vWFs to look like the inputs of vWFc. The idea is simple: the problem of POD is that it encodes in the modes the variations (in time and parameters) of the snapshots given in a hierarchical order: the first modes are the ones that enclose the biggest variations. In our case time is the parameter that gives a lot of variability both in concentration and in shape. In the case of vWFs, early times are characterized by weaker and very localized flows that appear for a small interval of time: to be correctly represented they require a big mixture of POD modes. To avoid this we need to create two different time windows as follows:

- Instead of having one snapshot matrix $\mathbf{S}^{N_h,N_s}$, we create two snapshot matrices $\mathbf{S}_1^{N_h,N_{s1}}$, $\mathbf{S}_2^{N_h,N_{s2}}$, where $N_{s1} = N_{train} \times N_{t_1}$ and $N_{s2} = N_{train} \times N_{t_2}$. In what follows N_{t_1} goes from $t = 2\,s$ to $t = 5\,s$ and N_{t_2} goes from $t = 6\,s$ to $t = 41\,s$. So $N_{t_1} = 4$ and $N_{t_2} = 36$.

- After the creation of the two snapshot matrices we perform the POD on both of them: in this way we have **two** different POD basis $\mathbf{V}_N^1$ and $\mathbf{V}_N^2$. N is the same since it refers to the size of the POD basis, the difference is that the

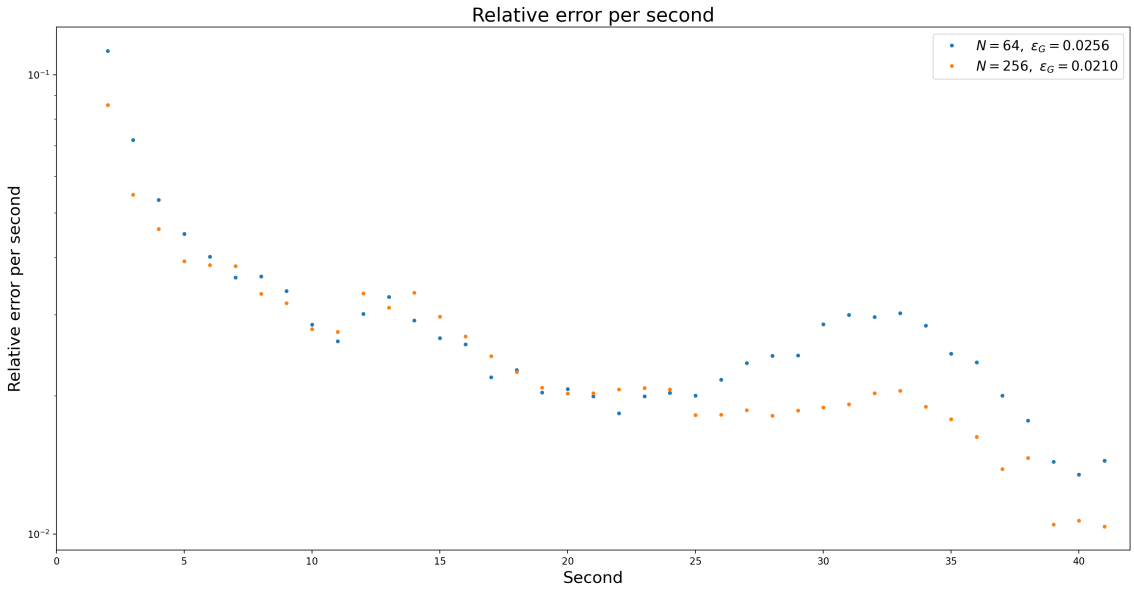

Figure 20: Error in time and global error of vWFc predictions for $N = 64, 256$.

modes of $\mathbf{V}_N^1$ can be used to predict only snapshots in the range $t = 2$ to $t = 5$, while the modes of $\mathbf{V}_N^2$ for the snapshots in the range $t = 6\,s$ to $t = 41\,s$.

- So following this distinction we create **one** input matrix $\mathbf{I}^{N,N_s}$, which has the same dimensions as in Section 3.3, but the coefficients are obtained projecting every snapshot onto the correct basis (depending on t).
- We now normalize $\mathbf{I}^{N,N_s}$, but we will look for $[max_1, min_1]$ and $[max_2, min_2]$, i.e. we have a max and a min for every time window, and we normalize according to Equation (18) (the snapshots from $t = 2$ to $t = 5$ normalized with $[max_1, min_1]$ and vice versa). The parameter matrix $\mathbf{M}$ instead, which is the input for the DFNN is constructed as usual.

Of course at testing time we will use one set of max, min or the other according to the time window in which the time we want to predict follows. We do not expect the fact of having coefficients belonging to different vector basis to confuse the deep neural network, as every 'class' of coefficients has associated its corresponding second t as input to the DFNN. In Figure 21 we can observe the results of the two time window approach: vWFs is now well predicted also at early times. Let us look at the relative error per second confronted with the previous method in Figure 22: there is a clear gain in the early times, but it performs worse for the last seconds, resulting in a slightly worse global error. Also it is worth noticing the 'jump' in the error from $t = 5\,s$ to $t = 6\,s$, which indicates the passage from base $\mathbf{V}_N^1$ to $\mathbf{V}_N^2$. The idea of having multiple **localized** reduced order models seems promising, since we do not want the POD to make us lose the information about the different regimes (in time and/or space) that a system can have. We are currently trying to make this idea more rigorous using clustering methods such as k-means [26] and the fuzzy c-means [27]. It is important to rely on robust methods to do the cut in time or parameters as it is not always possible to see a clear separation in physical regimes, especially if it is not in time but in some other parameter.

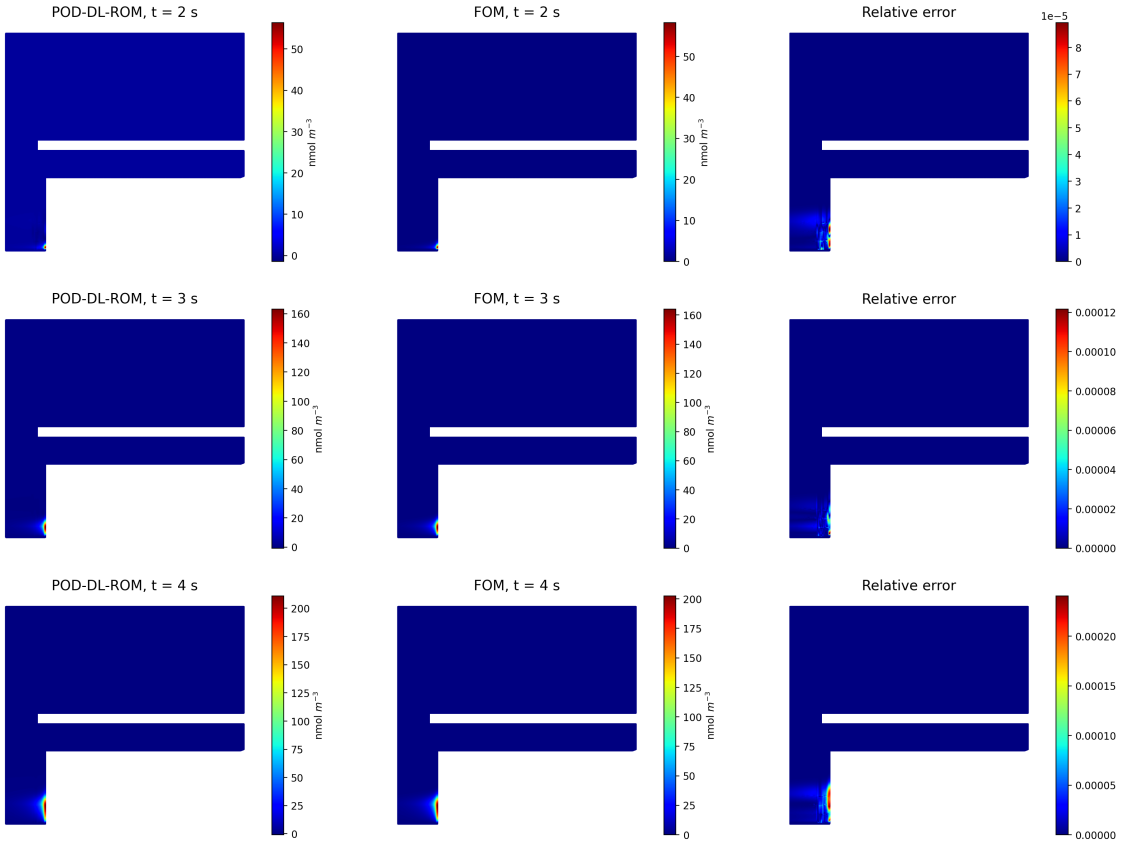

Figure 21: Predictions of vWFs with separation in time windows. $N = 64$

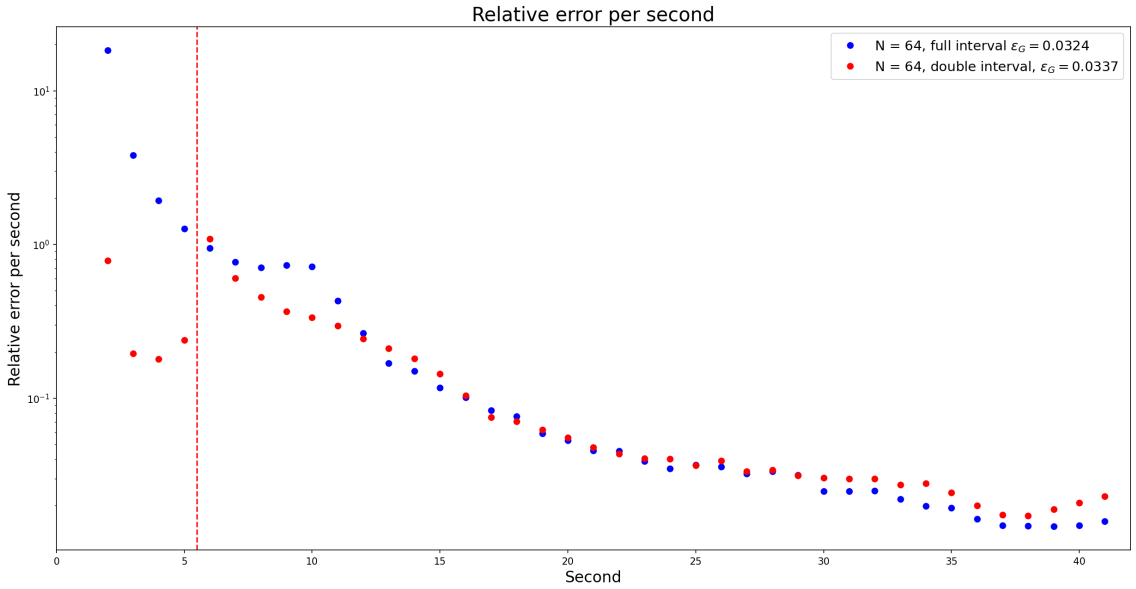

Figure 22: Comparison of relative and global error between full interval and two intervals methods for vWFs. $N = 64$ in both cases. The red line denotes where the cut in time for the two intervals is made.

4 Localized-POD-DL-ROM

On the wake of the insights gained in the previous section, we tried to make the **localized** reduced order model method quantitative, avoiding the need to decide empirically where to make the cut(s) in time and/or parameters. Taking inspiration from the work proposed in [28], we decided to rely on a **clustering** method in order to identify the regimes existing in time and/or parameters. This new procedure, which we will refer to as to the **Localized-POD-DL-ROM**, during the *training* stage is made up by 3 steps:

1. **Clustering step**: we divide the snapshots in k clusters, which will contribute to the construction of k snapshots matrices and thus k POD basis.

2. **Classification step**: We train a classification NN, which learns the mapping $(t, \boldsymbol{\mu}) \rightarrow x \in \boldsymbol{\chi}$, where $\boldsymbol{\chi} = \{0, 1, 2, ..., k-1\}$. In this way we can map an instance of parameters $(t, \boldsymbol{\mu})$ which has never been seen by the clustering onto the correct POD basis.

3. **POD-DL-ROM step**: As it is done in section 3.4 we construct the Input matrix $\mathbf{I}$, which will contain the POD coefficients obtained by projecting every snapshot onto one of the k POD basis to which it belongs, according to the clustering of step 1.

At the end of this *training* stage we have two NN models: a **NN Classifier** and a **Convolutional AutoEncoder** together with a **Deep FeedForward NN**. The latter has exactly the purpose of the previous sections, the former is needed since at testing time we need to know **to which cluster** k the parameter instance $(t, \boldsymbol{\mu})$ belongs. This means that during the *testing* stage we have the 3 following steps:

1. **DL-ROM step**: the testing parameter instance $(t, \boldsymbol{\mu})$ is given as input to the DFNN, and the Decoder gives as output a set of POD coefficients $\mathbf{u}_N(t, \boldsymbol{\mu})$.

2. **NN classifier step**: the same testing parameter instance $(t, \boldsymbol{\mu})$ is given as input to the NN Classifier, which will give as output the cluster to which the input belongs.

3. **Reconstruction step**: we recover the field $\mathbf{u}_h(t, \boldsymbol{\mu})$ by the approximation $\mathbf{u}_h(t, \boldsymbol{\mu}) \approx \mathbf{V}_i \mathbf{u}_N(t, \boldsymbol{\mu})$, where $\mathbf{V}_i$ is the POD basis obtained from the cluster of snapshots i (chosen in step 2).

We will now show in more detail the methods employed.

4.1 Clustering

Following the work done in [28], we decided to use the Fuzzy C-means clustering algorithm (FCM) [27]. We applied the FCM algorithm not directly to the snapshots $\mathbf{u}(t, \boldsymbol{\mu})$ but to their POD coefficients vectors $\mathbf{u}_N(t, \boldsymbol{\mu})$ for two reasons: first of all by going from a space of 10^5 dimensions to one of N dimensions ($N \approx 10^1$) we make the FCM computation cheaper. Secondly, it makes more sense to act on the

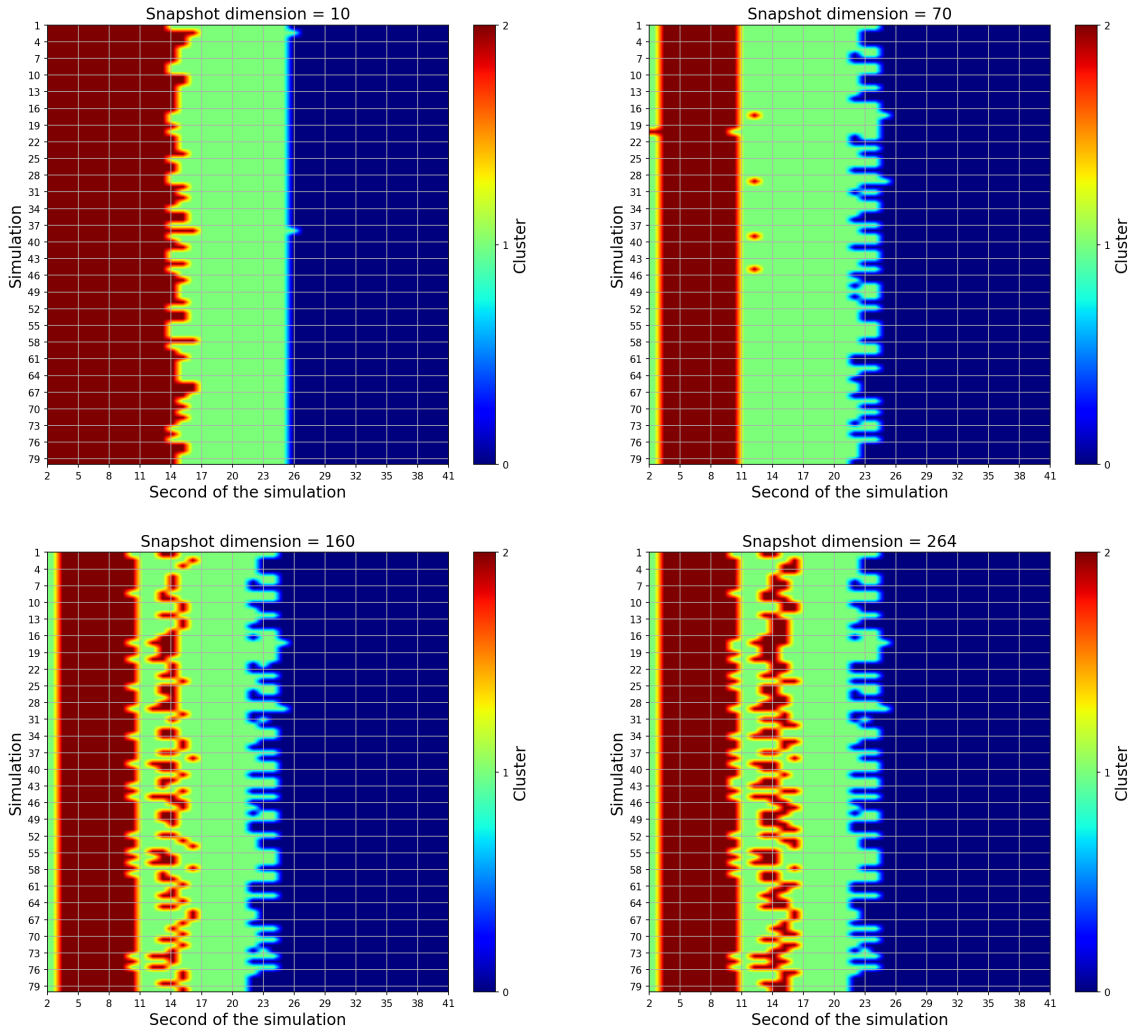

Figure 23: Clustering on POD coefficients for $k = 3$ at the variation of the dimension of the POD coefficients vectors (N).

POD coefficients vectors space instead of on the field space, since we are feeding the Encoder with the former. Be careful to make a distinction here: we are now doing a POD operation on **all** the snapshots in time and parameters as it is done in Section 3. This is needed for the two reasons above, but we will not use those POD coefficients for the DL-ROM procedure, they are just needed for the clustering step. We remind that the goal is to group in clusters **similar** POD coefficients vectors, since thanks to the previous sections we realized that when they are too different it creates difficulty in the DL-ROM procedure. After this step we will group in k clusters the snapshots (accordingly to how the corresponding POD coefficients vectors are grouped) and we will compute k POD basis. Thanks to those k POD basis we will get the **final** POD coefficients vectors which will feed the Encoder. The last technical point regards the **normalization** of the POD coefficients vector before implementing the clustering: since the clustering is used to group snapshots based on **which** POD modes are important for their reconstruction, we normalize every POD coefficients vector by their own maximum and minimum, by subtracting the minimum and dividing by the difference between maximum and minimum.

In Figure 23 we show the effect of the clustering at the variation of N with $k = 3$. This means that we are projecting the fields $\mathbf{u}_h(t, \boldsymbol{\mu})$ on a linear vector space of dimension N, with $N \in \{10, 70, 160, 264\}$ and we apply the FCM algorithm on POD coefficients vectors of dimension N. We chose $k = 3$ since we recognized 3 regimes in time for the previous sections.The first immediate comment about Figure 23 is that the clustering acts mainly on the space of time, meaning that time is the main parameter that causes variation in this dynamical system. In fact in all the 4 figures we see that given a second if we go through the simulation axis we do not see a change in clusters, except for the seconds near the two boundaries and some exceptions. Another comment is related to the difference between the case $N = 10$ and the others: first of all the snapshots with $t = 2\,s$ belong to the red cluster number 2, while they belong to cluster number 1 in the others. Secondly we see how two the cuts are shifted towards the left from $N = 10$ to the others. It is difficult to establish why all this changes happen when N becomes larger. For sure the snapshots with $t = 2\,s$ are much more different from the others, as we discussed in Figure 12, and this difference becomes clearer while N grows. We conclude this section by remarking that we used the FCM algorithms on 4000 POD coefficients vectors (40 seconds from 100 simulations). In what follows we choose the dimension of the POD coefficients vector on which it is done the **clustering** to be $N = 5$, since from Figure 23 it appears that with low N we get a more regular classification, especially for the snapshots with $t = 2\,s$.

4.2 NN classifier

The second step of the *training* stage regards the construction of a NN classifier, in what follows with $k = 3$. This object is needed since at testing time we will only have an instance of parameters $(t, \boldsymbol{\mu})$ and we will want to know **to which** POD basis it belongs. The Classifier is made up by 4 hidden layers of 50 neurons each. We use a ReLu activation function on all layers and a Softmax function on the last one. We train the network on 3200 samples, giving as input the $6-$dimensional vector $(t, \boldsymbol{\mu})$ and getting as output the corresponding cluster. The loss function used is the Sparse Categorical Cross Entropy. In Figure 24 we see the result of the application of the NN Classifier on the testing instance of parameters, with an accuracy of 0.9825. We used 3200 snapshots for the training and 800 for the testing. The prediction works well probably because time is the main driver for the grouping of clusters, and thus the model only needs to look at it. We conclude by explaining why we are training with 3200 samples while we used all the 4000 for the clustering: there is not danger of overfitting, since using all the data available for the clustering only makes it more accurate to **find the centers** of the k clusters, since the classification is later trained on the NN Classifier by using the first 3200 samples.

4.3 Results

After the clustering and classification step, we can finally use the POD-DL-ROM method as in Section 3. The architecture is the same discussed in 2.5.1. **The difference** now is that the Encoder is fed with POD coefficients vectors coming from

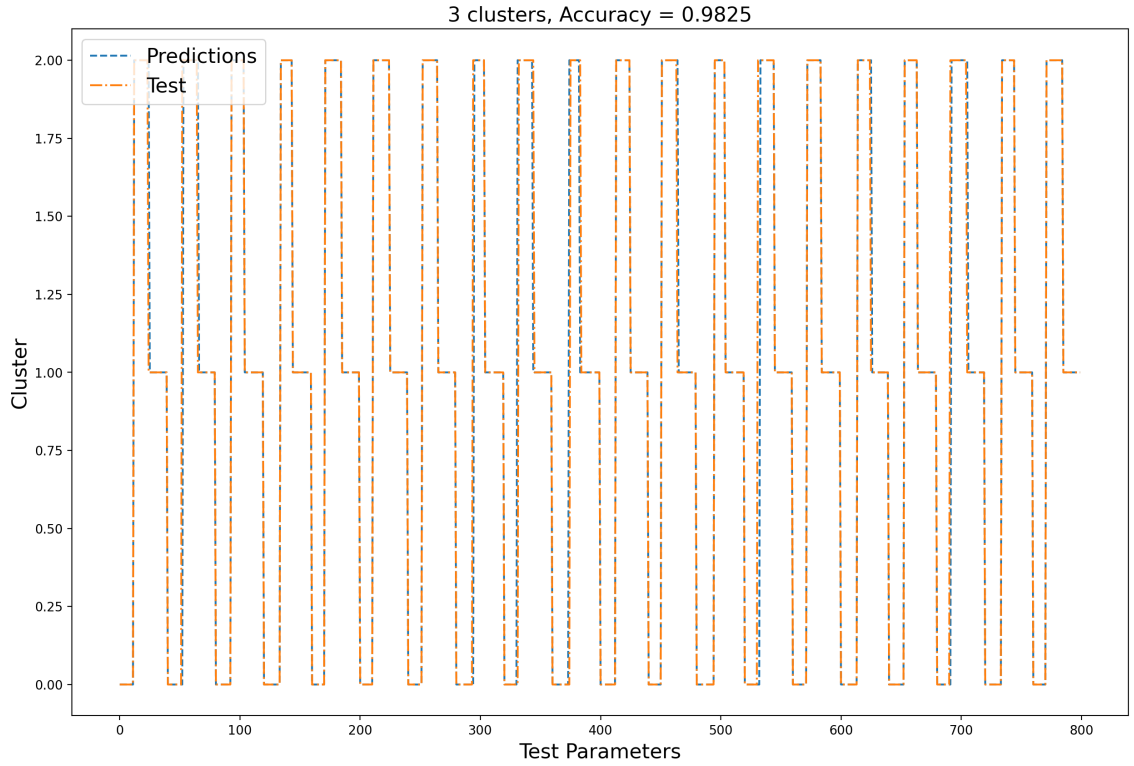

Figure 24: Predictions of the NN Classifier

k POD basis. At this regard, a small note about the normalization of the inputs: we proceed in the same way as in 3.1, but now we have $\mathbf{I}^1, \mathbf{I}^2, ..., \mathbf{I}^k$, coefficient matrices, and thus k $\mathbf{I}^i_{max}$ and $\mathbf{I}^i_{min}$. This means that we apply equation 18 paying attention to use the correct $\mathbf{I}^i_{max}$ and $\mathbf{I}^i_{min}$. Of course at testing time when we use the inverse normalization, we keep track of the right $\mathbf{I}^i_{max}$ and $\mathbf{I}^i_{min}$. In Figure 25 we see the relative error per second of the three methods compared: standard POD-DL-ROM, Empirical POD-DL-ROM with only one cut in time recognized empirically (section 3.4) and Localized-POD-DL-ROM with $k = 3$. We used $N = 64$. Notice how the standard POD-DL-ROM outperforms the other methods given the ϵ_G (Equation 21). However for the first seconds both the Empirical and the Localized methods give better results than the Standard POD-DL-ROM. We remind that with the Localized POD-DL-ROM method we have **an additional source of error**: the one caused by the NN Classifier. In fact since its accuracy is high but still not one, if the prediction of the cluster is wrong it assigns the POD coefficients to a wrong POD basis, causing a major increase in the relative error per second and in ϵ_G. This happens often towards the 'cuts', as it can be seen around second 26 of Figure 25. In fact if we look at the case $N = 10$ of Figure 23 (we recall here we used $N = 5$ for the clustering), we see the second cut is around $t = 26\,s$. Finally, it should be noticed that the global metric we use, that is Equation (21), gives **much more importance** to the snapshots with greater norms. This comes from the subtle **difference** between the denominators of Equations (21) and (22): the first one takes into consideration the sum of the squared norms of **all** the snapshots of a given testing simulations, while the latter only the squared norm of the snapshot

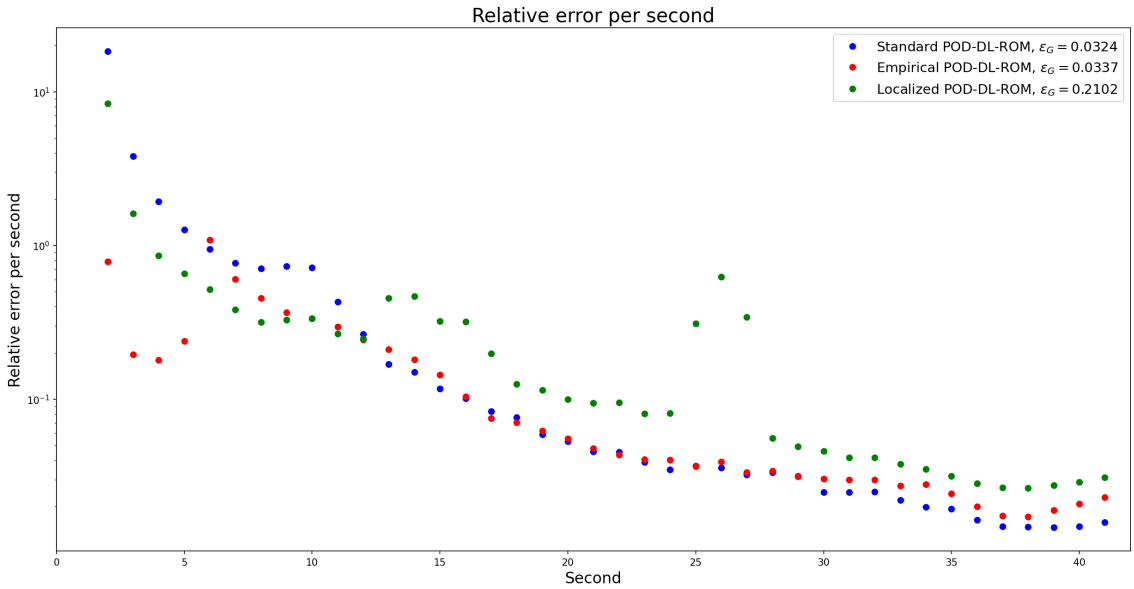

Figure 25: Comparison of the relative error per second of the 3 methods: Standard POD-DL-ROM, Empirical POD-DL-ROM and Localized POD-DL-ROM. $N = 64$ for all the 3 cases.

at time t. It follows that for the first metric it does not matter much how the predictions work for the initial seconds, since the norms of those vectors is small, compared to the late seconds. On the opposite the second metric only considers the norm of the snapshot of time t. This means that accordingly to ϵ_G, it is much more important to perform well towards the last seconds of the simulations than the initial ones. This is why following ϵ_G the standard POD-DL-ROM performs better than the empirical one as it is shown in Figure 25, although the initial seconds are much better predicted in the latter than the late seconds by the former.

4.4 Independent training

We conclude with a final experiment: we use the clustering with $k = 3$ to understand how the snapshots are grouped in time and then we set the boundaries: we make by hands cuts at $t = 14\,s$ and $t = 26\,s$. This means now we are **not** using a NN Classifier, we just use the clustering to decide where to put the cuts. Of course this is not very rigorous as in Figure 23 we see that the clusters are not exactly only in time, but especially at the boundaries we have some irregularities. Then we train **independently** three models: we create 3 Input matrices $\mathbf{I}^1, \mathbf{I}^2, \mathbf{I}^3$ based on the two cuts in time and we use them to train 3 independent POD-DL-ROM models. This is different from Empirical POD-DL-ROM (Section 3.4), where we created multiple POD basis but only one Input matrix, obtaining at the end only one POD-DL-ROM model. In Figure 26 we see how this method performs better (according to ϵ_G) than the others.

Overall, we see how the Localized POD-DL-ROM method performs poorly, compared to the others. This is due for sure to the error introduce by the NN Classifier and to the fact that the convolutional layers of the CAE are given some images

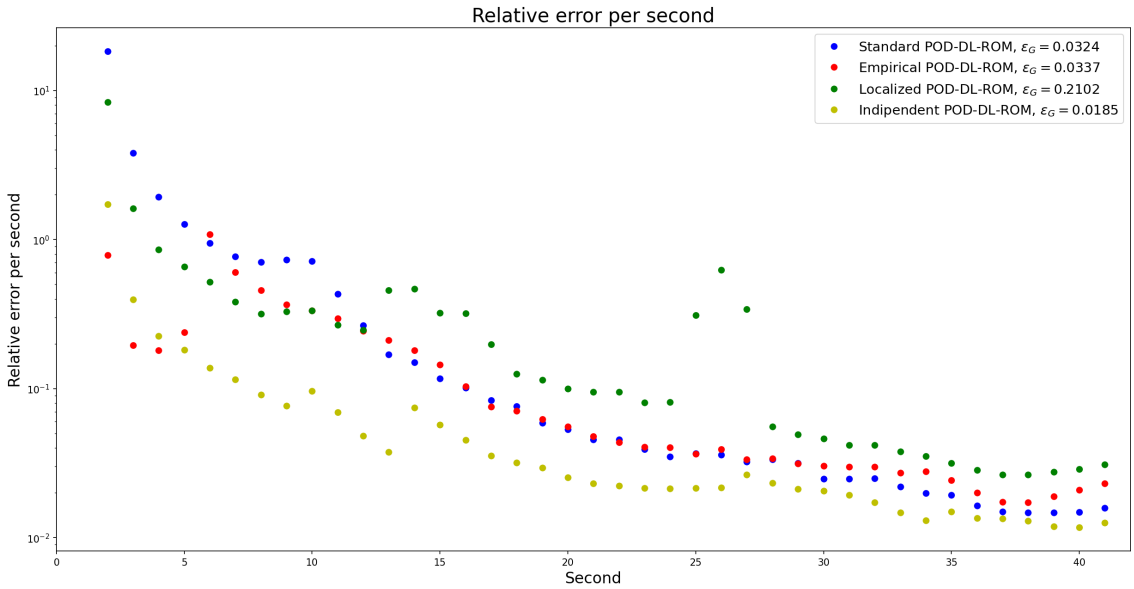

Figure 26: Comparison of the relative error per second of the 4 methods: Standard POD-DL-ROM, Empirical POD-DL-ROM and Localized POD-DL-ROM and Independent POD-DL-ROM . $N = 64$ for all the 4 cases.

which look similar but belong to different POD basis. However the experiment done with the independent training showed how the insight of using an **initial** clustering to separate the regimes may point in the right direction, although it is still not clear how to concretise everything in a unique NN model.

A final theoretical **open question**, related to the internal structure of the DL-ROM method: how is it theoretically justifiable to use convolutional layers on POD coefficients vector, when these are independent? Convolutional layers are much effective, for example in image classification, since they learn the spatial correlations internal to objects, but this should not be the case in our system, where the Encoder inputs are POD coefficients. This topic is partially addressed in the paper [29], in which the very strong analogy between the DL-ROM / POD-DL-ROM techniques and the DPIM method is studied. Because of this strong similarity the authors are led to say that in POD-DL-ROM the POD modes are combined with each other through nonlinear functions of the latent variables. However this still conflicts with the independence of the POD modes, and when the presence of POD coefficients from different POD basis is added, this becomes even more difficult to explain on a theoretical basis. It would be interesting to see how the performance of POD-DL-ROM would change if the convolutional layers were replaced by dense layers, although the training would be slower.

5 Conclusion

In this work we showed how to construct a reduced order model of blood **thrombosis** simulations using the POD-DL-ROM method. We focused on the prediction in time and space of one biochemical specie, vWFs, which is chosen since it is the one that makes the prediction difficult for some time instances. We were able to get good results at the variation of both time and parameters, with the exception of some cases that we presented. We found out that, although POD is an essential tool to reduce the dimensionality of the problem, it poses a huge conceptual limitation to the objective of the work: it limits the approximation of the fields to a linear combination of vectors, which is non-ideal when the PDEs are nonlinear. It is true that this problem may be partially solved increasing the size of the linear basis, but as we showed in Figure 17, this implies a bigger number of parameters to be learned by the algorithm and thus a bigger number of data-points for the training. In addition, POD modes favor the variations in time and parameters which carry the biggest amount of energy, and thus it is difficult to find the correct linear combination of modes which can result in a good approximation of the field. To overcome this limit we showed in Section 3.4 an example of (empirical) **localized** model order reductions. We separated the snapshots in two time intervals, but the separation in general should not be limited to time: there may be systems characterized by different regimes not at the variation of time but of some other parameter. For this reason we introduced the Localized POD-DL-ROM, which relies on a prior clustering step in order to find the different regimes in time and/or parameters in a quantitative (non empirical) way. However this method does not lead to better results when compared to the others proposed. It would however be worth it to try to improve it, because of its non-intrusive nature and its potentiality to deal with big variations in time and/or parameters inside a physical simulation. To confirm the usefulness of an **initial** clustering, we showed how training k **independent** POD-DL-ROM models from snapshots divided by the clustering outperforms the traditional POD-DL-ROM, although in this way we renounce to a single NN model. Future lines of research may also point in the following direction: finding a substitute of POD which is a linear approximation, by exploring nonlinear dimensionality reductions such as KPOD [30].

Source code: the code for POD-DL-ROM and Localized POD-DL-ROM is available at
https://github.com/Aleartulon/rom4clot

References

[1] P. Nagareddy and S. Smyth, "Inflammation and thrombosis in cardiovascular disease. current opinion in hematology," 2013.

[2] Jaffer and et al., "Acta biomater," 2019.

[3] R. M. Rojano, D. Lucor, and et al., "Uncertainty quantification of a thrombosis model considering the clotting assay pfa-100®," 2021.

[4] T. Lassila, A. Manzoni, A. Quarteroni, and G. Rozza, "Model order reduction in fluid dynamics: Challenges and perspectives," pp. 235–273, 2014.

[5] P. Harrison, M. Robinson, R. Liesner, and et al., "A potential rapid screening tool for the assessment of platelet dysfunction," *Clinical and Laboratory Haematology*, 2002.

[6] M. Zhussupbekov, R. Méndez Rojano, W.-T. Wu, and J. Antaki, "Von willebrand factor unfolding mediates platelet deposition in a model of high-shear thrombosis," 2022.

[7] P. Zhang, J. Sheriff, S. Einav, and et al., "A predictive multiscale model for simulating flow-induced platelet activation: Correlating in silico results with in vitro results.," 2021.

[8] W. Wu, M. Jamiolkowski, W. Wagner, and et al., "Multi-constituent simulation of thrombus deposition.," 2017.

[9] P. Benner, M. Ohlberger, A. Cohen, and K. Willcox, *Model reduction and approximation: theory and algorithms*. SIAM, 2017.

[10] M. Ohlberger and S. Rave, "Reduced basis methods: Success, limitations and future challenges," *arXiv: Numerical Analysis*, 2015.

[11] K. Lee and K. T. Carlberg, "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders," *Journal of Computational Physics*, 2020.

[12] S. Fresca, L. Dede, and A. Manzoni, "A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes," *Journal of Scientific Computing*, 2021.

[13] S. Fresca and A. Manzoni, "Pod-dl-rom: Enhancing deep learning-based reduced order models for nonlinear parametrized pdes by proper orthogonal decomposition," *Computer Methods in Applied Mechanics and Engineering*, 2022.

[14] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM Review*, 2015.

[15] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual Review of Fluid Mechanics*, 1993.

[16] L. Sirovich, "Turbulence and the dynamics of coherent structures. i - coherent structures. ii - symmetries and transformations. iii - dynamics and scaling," *Quarterly of Applied Mathematics*, 1987.

[17] M. Rathinam and L. Petzold, "A new look at proper orthogonal decomposition," *SIAM J. Numerical Analysis*, 2003.

[18] CNRS, "Formation introduction au deep learning," Available at https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/-/wikis/home.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Adaptive computation and machine learning, MIT Press, 2016.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," (Red Hook, NY, USA), Curran Associates Inc., 2012.

[21] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," *Physics Reports*, 2019.

[22] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1993.

[23] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective," *IEEE Transactions on Cybernetics*, 2020.

[24] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, 2011.

[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[26] D. Pelleg and A. Moore, "Accelerating exact k-means algorithms with geometric reasoning," 1999.

[27] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," 1984.

[28] R. Geelen and K. Willcox, "Localized non-intrusive reduced-order modelling in the operator inference framework," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2022.

[29] G. Gobat, S. Fresca, A. Manzoni, and A. Frangi, "Virtual twins of nonlinear vibrating multiphysics microstructures: physics-based versus deep learning-based approaches," 05 2022.

[30] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, 1998.