http://www.aimspress.com/journal/mine

Mathematics in Engineering, 5(6): 1–36.
DOI:10.3934/mine.2023096
Received: 30 November 2022
Revised: 04 May 2023
Accepted: 14 July 2023
Published: 05 September 2023

Research article

Long-time prediction of nonlinear parametrized dynamical systems by deep learning-based reduced order models

Stefania Fresca, Federico Fatone and Andrea Manzoni[*]

MOX–Dipartimento di Matematica, Politecnico di Milano, P.zza Leonardo da Vinci 32, 20133 Milano, Italy

[*] **Correspondence:** Email: andrea1.manzoni@polimi.it.

Abstract: Deep learning-based reduced order models (DL-ROMs) have been recently proposed to overcome common limitations shared by conventional ROMs–built, e.g., through proper orthogonal decomposition (POD)–when applied to nonlinear time-dependent parametrized PDEs. In particular, POD-DL-ROMs can achieve an extremely good efficiency in the training stage and faster than real-time performances at testing, thanks to a prior dimensionality reduction through POD and a DL-based prediction framework. Nonetheless, they share with conventional ROMs unsatisfactory performances regarding time extrapolation tasks. This work aims at taking a further step towards the use of DL algorithms for the efficient approximation of parametrized PDEs by introducing the μt-POD-LSTM-ROM framework. This latter extends the POD-DL-ROMs by adding a two-fold architecture taking advantage of long short-term memory (LSTM) cells, ultimately allowing long-term prediction of complex systems' evolution, with respect to the training window, for unseen input parameter values. Numerical results show that μt-POD-LSTM-ROMs enable the extrapolation for time windows up to 15 times larger than the training time interval, also achieving better performances at testing than POD-DL-ROMs.

Keywords: reduced order modeling; deep learning; proper orthogonal decomposition; long-short term memory networks; time forecasting; parametrized PDEs

1. Introduction

Parameterized PDEs are extensively used for the mathematical description of several physical phenomena. Some instances include fluid dynamics, heat transfer, waves and signal propagation and structure dynamics (including microsystems) [1–3]. However, traditional high-fidelity, full order models (FOMs) employed for their numerical approximation, such as those based on the finite element method, become infeasible when dealing with complex systems and multiple input-output responses need to be evaluated (like, e.g., for uncertainty quantification, control and optimization) or real-time performances must be achieved. In fact, despite being accurate up to a desired tolerance, they entail unaffordable computational times, sometimes even orders of magnitude higher than the ones required by real-time computing [4].

In this context, projection-based reduced order models (ROMs)–such as POD-Galerkin ROMs–have been introduced with the goal of enhancing efficiency in timing critical applications. ROMs rely on a suitable offline-online computational splitting, aimed at moving in the offline stage, i.e., the phase where the model is trained and refined before its deployment, the computationally expensive tasks in order to make the online one, i.e., the phase where the model is used for the solution of the problem for new parametric instances, extremely efficient. These methods rely on the assumption that the parameterized PDE solutions manifold can be represented by the span of a small number of basis functions built starting from a set of FOM solutions (computed in the offline stage), the so-called *reduced manifold*. This pipeline allows for a significant dimensionality reduction of the PDE problem and a consequent speed-up in its numerical solution timing, for example a $O(10^2)$ speed-up is achieved for Navier-Stokes application [5–7], and even more for applications in structural mechanics [8, 9].

Nevertheless, despite being physics-driven, POD-Galerkin ROMs show severe limitations when addressing nonlinear time-dependent PDEs, which might be related to *(i)* the need to rely on high-dimensional linear approximating trial manifolds, *(ii)* the need to perform expensive hyper-reduction strategies, or *(iii)* the intrinsic difficulty to handle complex physical systems with a linear superimposition of modes [2]. Furthermore, usually such ROMs do not allow for an effective extrapolation in time when dealing with time- and parameter-dependent problems–they can make predictions beyond the training time range in the case of time-dependent linear (or mildly nonlinear) systems not including any parameter dependence, requiring extremely long offline stages in order to compute FOM snapshots defined on a sufficiently long time domain.

To overcome these drawbacks, several nonlinear–and in particular artificial neural networks (ANNs) based–methods have been massively considered to provide fast approximation of PDE solutions in the last few years, and even before. For instance, the possibility to approximate differential equations solutions through ANNs had already been proposed in [10, 11], relying on the universal approximation theorem [12]. Interest on the topic and practical applications increased recently at a fast pace [13–15], while variations aimed at introducing physics related losses to link more deeply the ANN framework with the underlying physical model, e.g., with the concept of physics-informed neural networks (PINNs) [16, 17], show the significance ANNs are gaining in scientific computing. Furthermore, relevant theoretical results concerning the complexity bounds of the neural network architectures for the approximation of either continuous and discretized functions or operators have also been investigated, see, e.g., [18–21] and [22–24] respectively, thus providing a rigorous mathematical setting to ANNs based methods.

The combination of the extremely accurate approximation capabilities of ANNs [18, 20] and the ROM paradigm led to the introduction of ANN-based ROMs, whose main idea is to use deep learning (DL) algorithms to perform a nonlinear projection onto a suitable reduced order manifold. For instance, in [25–27] a DL-based regressor is employed but a linear reduced manifold is still considered. In [28], an ANN-inferred correction term is used to increase the accuracy of the linear projection, while [29, 30] approximate the reduced manifold by means of an ANN. In [31], a convolutional autoencoder is considered to model the reduced order manifold, however advancing in time through a quasi-Newton method that requires the approximation of a Jacobian matrix at every time step.

A recently proposed strategy [32] aims at constructing DL-based ROMs (DL-ROMs) for nonlinear time-dependent parametrized PDEs in a non-intrusive way. DL-ROM aims at approximating both the PDE solution manifold, that is the set of all PDE solutions by varying time and parameters, by means of a low-dimensional, nonlinear trial manifold, and the nonlinear dynamics of the intrinsic coordinates on such trial manifold, as a function of the time coordinate and the parameters. The approximated manifold is learnt by means of the decoder function of a convolutional autoencoder (CAE) neural network; the reduced dynamics through a (deep) feedforward neural network (FFNN), and the encoder function of the CAE. DL-ROMs outperform projection-based ROMs such as the reduced basis method–regarding both numerical accuracy and computational efficiency at testing stage. With the same spirit, POD-DL-ROMs [33] enable a more efficient training stage and the use of much larger FOM dimensions, without affecting network complexity. This is achieved by means of a prior dimensionality reduction of FOM snapshots through randomized POD (rPOD) [34], and a multi-fidelity pretraining stage, where different models (exploiting, e.g., coarser discretizations or simplified physical models) can be combined to iteratively initialize network parameters. The POD-DL-ROM technique has proven to be effective for instance in the real-time approximation of cardiac electrophysiology problems [35, 36], problems in fluid dynamics [37] and Micro-Electro-Mechanical Systems application [38].

This work extends the POD-DL-ROM framework [33] in two directions: first, it replaces the CAE architecture of a POD-DL-ROM with a long short-term memory (LSTM) based autoencoder [39, 40], in order to better take into account time evolution when dealing with nonlinear unsteady parametrized PDEs (μ-POD-LSTM-ROM); second, it aims at performing extrapolation forward in time (compared to the training time window) of the PDE solution, for unseen values of the input parameters–a task often missed by traditional projection-based ROMs. Several works have tried to account for temporal correlations between consequent time instances. For example, in [41] a time-stepping procedure based on the use of residual neural networks is presented but no extrapolation forward in time is performed for the test cases reported, as in [42]. Instead, other contributions have focused on the temporal advancement of the approximated solution on a time window larger with respect to the one seen during the training phase, such as in [43, 44], where either causal convolutions and LSTMs, or a three levels neural network are employed for future time steps predictions.

Our final goal is to predict the PDE solution on a larger time domain (T_{in}, T_{end}) than the one, $(0, T)$, used for the ROM training–here $0 \leq T_{in} \leq T_{end}$ and $T_{end} > T$. To this aim, we train a μ-POD-LSTM-ROM using N_t time instances and approximate the solution up to $N_t + M$ time steps from the starting point, taking advantage of a time series LSTM-based architecture (t-POD-LSTM-ROM) besides the μ-POD-LSTM-ROM introduced before. These architectures mimic the behavior of numerical solvers as they build predictions for future times based on the past. Besides this, the implications of the novelties proposed by the present work are multiple. In particular, the main advantages concern:

- The possible long-term time extrapolation capabilities of the proposed framework, allowing for a faster offline stage, as FOM snapshots defined on a shorter time domain are required to train the model;
- The possibility to predict entire sequences instead of single outputs, that makes the presented method even more efficient than the (already faster than real-time) POD-DL-ROMs [33],

while at the same time preserving the main strengths of POD-DL-ROMs, which are:

- The possibility to query the ROM at a specific time instance, without requiring the solution of a dynamical system until that time as usual time marching schemes would do;
- The possibility of using coarser temporal discretizations with respect to the ones used to ensure stability for high-fidelity numerical solvers [35];
- The avoidance of using expensive hyper-reduction techniques often required by POD-Galerkin ROMs;
- The possibility to return outputs depending on selected problem state variables, without requiring to approximate all of them.

The paper is divided in five sections: In Section 2 we describe the μt-POD-LSTM-ROM framework used to predict PDE solutions for unseen parameter instances and times. Section 3 and Section 4 introduce the μ-POD-LSTM-ROM and t-POD-LSTM-ROM architectures, the former enriching POD-DL-ROM with LSTM-based autoencoder and the latter providing time extrapolation capabilities to the framework. In Section 5 we report the accuracy results and performances assessments of μt-POD-LSTM-ROM on three parametrized test cases, namely: *(i)* 3 species Lotka-Volterra equations, *(ii)* unsteady advection-diffusion-reaction equation, *(iii)* incompressible Navier-Stokes equations.

2. Achieving time extrapolation capabilities with LSTM cells

After recalling the formulation of a POD-DL-ROM, in this section we address the construction of the proposed μt-POD-LSTM-ROM framework to predict PDE solutions for unseen parameter instances and times; the main ingredients to reach this goal–the μ-POD-LSTM-ROM and the t-POD-LSTM-ROM architectures–will instead be detailed in the following sections. The space and time discretization on a nonlinear, time-dependent, parametrized PDE problem–performed, e.g., through a finite element method–produces a (high-dimensional) dynamical system of the form:

$$\begin{cases}\mathbf{M}(\boldsymbol{\mu})\dot{\mathbf{u}}_h(t;\boldsymbol{\mu}) = \mathbf{f}(t,\mathbf{u}_h(t;\boldsymbol{\mu});\boldsymbol{\mu}), & t \in (0,T), \\ \mathbf{u}_h(0;\boldsymbol{\mu}) = \mathbf{u}_0(\boldsymbol{\mu}). \end{cases} \tag{2.1}$$

where $\mathbf{u}_h : (0,T)\times\mathcal{P} \to \mathbb{R}^{N_h}$ is the parametrized solution of (2.1), $\mathbf{u}_0 : \mathcal{P} \to \mathbb{R}^{N_h}$ is the initial datum, $\mathbf{f} : (0,T)\times\mathbb{R}^{N_h}\times\mathcal{P} \to \mathbb{R}^{N_h}$ is a (nonlinear) function, representing the system dynamics and $\mathbf{M}(\boldsymbol{\mu}) \in \mathbb{R}^{N_h\times N_h}$ is the mass matrix of this parametric FOM, assumed here to be a symmetric positive definite matrix. Here N_h and $\mathcal{P} \subset \mathbb{R}^{n_\mu}$ denote the FOM dimension and the parameters' space, respectively.

Linear projection-based ROMs exploit singular value decomposition (SVD) of the FOM snapshot matrix $\mathbf{S}$, i.e., the data structure containing the full order solutions (*snapshots*) used for training, in order to build a N-dimensional space basis and project the system (2.1) on it. In this way, an N-dimensional reduced solution manifold $\mathcal{M}_N$ is obtained [2]. Since solving the FOM (2.1) can be computationally unaffordable, we aim at replacing it with the approximation obtained through suitable ROMs.

Since (2.1) entails a temporal evolution of the described phenomenon, the ROMs used to approximate its dynamics must include some kind of time parameter (even dimensionless), or at least some procedure allowing the advancement in time of the solution to work properly. POD-Galerkin ROMs, for instance, usually consider a time marching scheme to handle the dynamical system they entail, similarly to the ones used at the FOM level (e.g., finite differences or backward differentiation formulas). POD-DL-ROMs as described in [33] treat instead the time like an additional physical parameter to be provided as input to the feedforward neural network $\boldsymbol{\phi}_n^{FFNN}(\cdot;\cdot,\boldsymbol{\theta}_{FFNN})$–being $\boldsymbol{\theta}_{FFNN}$ its trainable parameters. This function maps the parameter vector $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}$ to the low (n-)dimensional nonlinear trial manifold, approximation of the solution manifold, i.e. the set of all the PDE solution by varying time and parameters, where n is very close or even equal to the intrinsic dimension of the problem $(n_\mu + 1)$:

$$\mathbf{u}_n(t;\boldsymbol{\mu}) = \boldsymbol{\phi}_n^{FFNN}(t;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN}),$$

so that in the end, the network learns a mapping of the form

$$\boldsymbol{\phi}_n^{FFNN}(\cdot;\cdot,\boldsymbol{\theta}_{FFNN}) : (0,T)\times\mathbb{R}^{n_\mu} \to \mathbb{R}^n.$$

In this way, starting from each pair $(time, parameters)$, $\boldsymbol{\phi}_n^{FFNN}(\cdot;\cdot,\boldsymbol{\theta}_{FFNN})$ produces the low-dimensional representation of the solution for those particular instances with a direct input-output relation for each time instant. Despite being fast and accurate, this approach neglects the correlation between consequent time steps of the solution, leaving the opportunity to further increase its efficiency. In fact, the simple selection of an initial condition and of an initial time should ideally contain enough information to reconstruct the entire temporal evolution of the solution. In this context, it would be ideally possible–and desirable–to obtain a map under the form

$$\boldsymbol{\Lambda}_N(\cdot,\boldsymbol{\theta}_\Lambda) : \mathbb{R}^{n_\mu} \to \mathbb{R}^N \times (0,T)$$

that, considering the initial time as $t = 0$, would provide the solution for a time horizon $(0,T)$ as long as necessary, thus enhancing time extrapolation capabilities. In this case, time would be considered implicitly by the model; this latter shall then learn the evolution of the problem through its trainable parameters. This would entail the setting of an algorithm–conceptually closer to a classical numerical solver than what presented before–as the obtained solution would be a sequence of vectors representing the evolution of the system in time, rather than a single result of a specific time query.

To better fit the working mechanism of numerical solvers, the application of recurrence strategies to ANNs emerges as a suitable solution, as they add to traditional feedforward architectures feedback connections allowing to treat inputs and outputs in the form of sequences. This is expected to enhance the reconstruction of the underlying dynamics, as this latter can be learned by the network implicitly.

Moreover, such architectures have proven to be effective in time series prediction problems [45]–even in the context of PDEs approximation [46]–opening the possibility for ROMs to advance in time with respect to the FOM snapshots they are trained with, performing extrapolation in time. Finally, recurrence mechanisms such as LSTM are also suitable for increasing speed performances at prediction time, as entire long temporal sequences can be returned as output from the architecture, requiring less neural network queries, and thus improving the overall efficiency of the method.

2.1. Time extrapolation problem

In the context of ROMs, the problem of time extrapolation requires the training of a ROM (eventually based on deep learning) on a training set including snapshots

$$\mathbf{u}_h(t;\boldsymbol{\mu}) \quad \text{with } \boldsymbol{\mu} \in \mathcal{P}_{train} \text{ and } t \in (0,T) \tag{2.2}$$

to be used to predict solutions defined on a larger temporal domain $\mathcal{T} = (T_{in}, T_{fin})$ with $0 \le T_{in} \le T_{fin}$ and $T_{fin} > T$:

$$\mathbf{u}_h(t;\boldsymbol{\mu}) \quad \text{with} \;\; \boldsymbol{\mu} \in \mathcal{P}_{test} \text{ and } t \in \mathcal{T}. \tag{2.3}$$

Parameters $\mathcal{P}_{test} \subset \mathcal{P}$ and $\mathcal{P}_{train} \subset \mathcal{P}$ are such that $\mu_i^{train,min} \leq \mu_i^{test} \leq \mu_i^{train,max} \;\; \forall i \in \{1, \ldots, n_\mu\}$.

In this case, an analogy with time marching numerical solvers can be found. Indeed, traditional numerical solvers integrate in time the system of PDEs starting from the given initial condition $\mathbf{u}_h(0;\boldsymbol{\mu})$, and build the solution iteratively exploiting past solution's values found by the solver itself. Our goal is to build a DL-based solution *approximator* able to proceed in time in a similar fashion. Ultimately, our objective is to train the framework using (the first) N_t time steps of the FOM solution, to get the solution of the problem up to $N_t + M$ time steps from the starting point.

2.2. *μt-POD-LSTM-ROM architecture*

The problem addressed in this paper is therefore two-fold: it deals with *(i)* the prediction of the solution of the parametric PDE problem for a new instance of the parameters' space belonging to the set $\mathcal{P}_{test} \subset \mathcal{P} \in \mathbb{R}^{n_\mu}$ (solution inference for new parameters values) and *(ii)* the forecast of the temporal evolution of that solution for unseen times (time extrapolation). A natural way to tackle this problem is to pursue a *divide-and-conquer* strategy, splitting its solution into a two-steps process that exploits two paired LSTM-based ANN architectures: the first one (μ-POD-LSTM-ROM) addressing the issue of predicting the solution for unseen parameters; the second one (t-POD-LSTM-ROM, where "t" stands for *time series*) extending the solution in time, starting from the sequence predicted by the former. We note that the t-POD-LSTM-ROM could, in principle, be combined with other parameter-dependent architectures or reduced order modeling framework.

The resulting technique (μt-POD-LSTM-ROM) can be described as follows (see Figure 1):

- The training stage is performed in parallel for the two paired architectures on the same dataset obtained from FOM solutions after a first POD-based dimensionality reduction. In the end, the μ-POD-LSTM-ROM will produce a structure able to predict the solution for unseen parameters, but on the same time domain considered during training, while the t-POD-LSTM-ROM will produce a time series predictor $\boldsymbol{\Lambda}_p(\cdot)$ that takes p time steps from the past and $\boldsymbol{\mu}$, and returns as output the forecast for k time steps in the future for that $\boldsymbol{\mu}$ value. From now on, the two architectures will act as separate entities;
- The μ-POD-LSTM-ROM takes the vector $(t_i, \boldsymbol{\mu})$ of a starting time in the interval $(0, T)$ (discretized in $\{t_0, \ldots, t_{N_t-1}\}$ time istances which are collected in subsequences) and of the parameters' instance, and performs the approximation of the solution on this time domain seen during the training stage. This step produces very accurate outputs that are also smoother in the time variable w.r.t. POD-DL-ROM ones thanks to the LSTM architecture producing sequences as outputs and not treating different time instances as independent. This enhances the performances of the time series predictor, as it would potentially incur stability issues by propagating the small oscillations somehow unavoidable in the POD-DL-ROM framework without LSTM cells. Note that all the predictions at this stage are performed in the reduced dimension N (the POD basis one);
- The t-POD-LSTM-ROM takes the last p time steps of the μ-POD-LSTM-ROM predicted sequence and advances another k steps. Then, it takes the last p time steps of the new predicted sequence and advances of other k steps, and keeps advancing the prediction in this way. This strategy thus performs extrapolation in time, virtually with no final time limit, acting as an auto-regressive model. Note that the t-POD-LSTM-ROM architecture is general enough to be used

also on top of other ROMs–e.g., POD-DL-ROMs or POD-Galerkin ROMs–in order to provide time extrapolation.

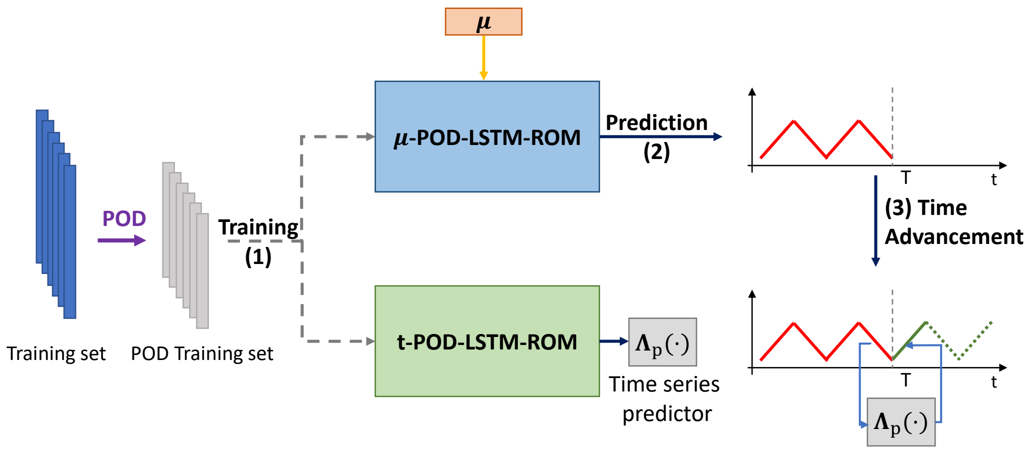

Figure 1. The μt-POD-LSTM-ROM framework. *(1) Training:* Both μ-POD-LSTM-ROM and t-POD-LSTM-ROM are trained on the same set of FOM snapshots reduced by means of (r)POD; *(2) Prediction:* The μ-POD-LSTM-ROM is employed to predict the (r)POD coordinates on the time interval $(0, T)$ on which the training snapshots were defined for new parameters instances; *(3) Time advancement:* Starting from the sequence predicted by the μ-POD-LSTM-ROM, the t-POD-LSTM-ROM time series predictor is used to advance in time and perform time extrapolation.

3. μ-POD-LSTM-ROM

The first component of the μt-POD-LSTM-ROM framework is a μ-POD-LSTM-ROM, originating from the application of a LSTM autoencoder structure in the context of POD-DL-ROMs. While a POD-DL-ROM aims at reducing the dimensionality of the solution by means of a nonlinear projection onto a suitable subspace, the proposed μ-POD-LSTM-ROM framework focuses on the compression of the information necessary to build an entire sequence of solutions. In particular, a LSTM autoencoder [47] takes a set of sequential inputs and through a LSTM architecture, the *encoder*, provides a lower dimensional representation of the entire sequence as a single vector. Another LSTM based ANN, the *decoder*, takes as input the aforementioned compressed representation and reconstructs the sequence of solutions used to produce it. Hence, such an autoencoder provides a convenient way of representing a sequence of solutions by compressing it in a much lower dimensional single vector, that can be inferred by a properly trained (ANN-based) regressor.

The structure of the μ-POD-LSTM-ROM described above (see Figure 2) starts with a first dimensionality reduction through the projection of the snapshots onto the POD basis as in a POD-DL-ROM. Snapshots are then put sequentially together in batches and–optionally–further reduced via some dense layers. Then, they are passed as sequences to the LSTM encoder architecture and the reconstruction process follows symmetrically through the decoder. At the same time, a feedforward neural network is set up for inferring the hidden representation of the LSTM autoencoder and, from that, for reconstructing the solution sequence starting from the tuple $(\boldsymbol{\mu}, t_i)$[*].

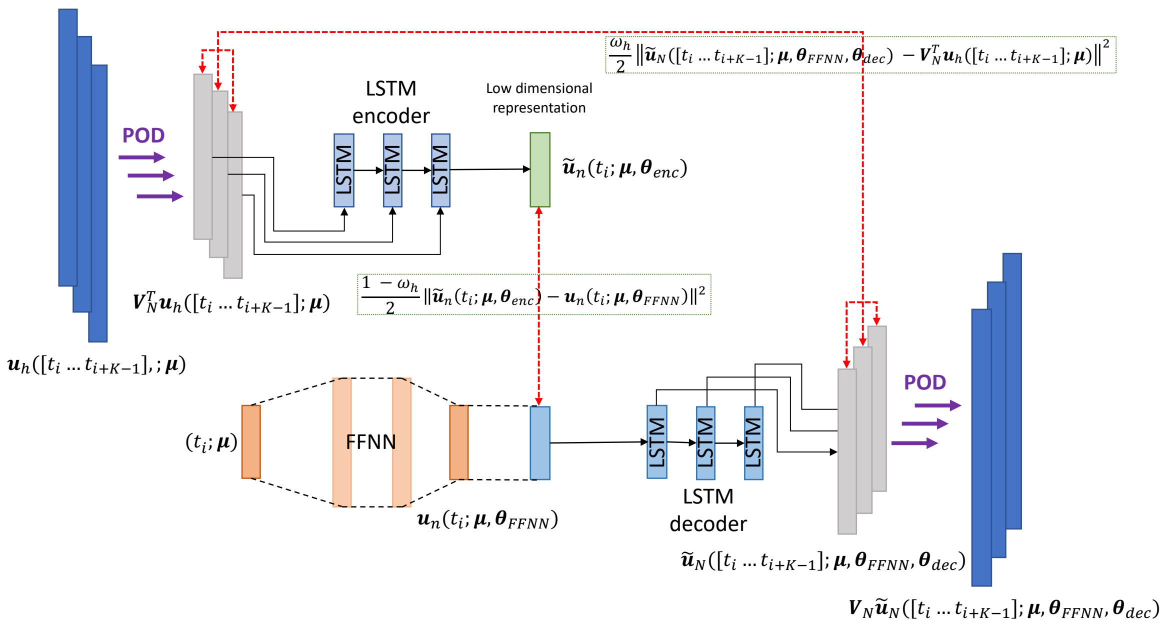
POD
LSTM encoder
Low dimensional representation
LSTM
$\frac{\omega_h}{2}\|\tilde{\boldsymbol{u}}_N([t_i \dots t_{i+K-1}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{dec}) - \boldsymbol{V}_N^T\boldsymbol{u}_h([t_i \dots t_{i+K-1}];\boldsymbol{\mu})\|^2$
$\tilde{\boldsymbol{u}}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc})$
$\boldsymbol{V}_N^T\boldsymbol{u}_h([t_i \dots t_{i+K-1}];\boldsymbol{\mu})$
$\frac{1-\omega_h}{2}\|\tilde{\boldsymbol{u}}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc}) - \boldsymbol{u}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN})\|^2$
$\boldsymbol{u}_h([t_i \dots t_{i+K-1}],;\boldsymbol{\mu})$
$(t_i;\boldsymbol{\mu})$
FFNN
$\boldsymbol{u}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN})$
LSTM decoder
POD
$\tilde{\boldsymbol{u}}_N([t_i \dots t_{i+K-1}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{dec})$
$\boldsymbol{V}_N\tilde{\boldsymbol{u}}_N([t_i \dots t_{i+K-1}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{dec})$

Figure 2. The μ-POD-LSTM-ROM architecture. The full order vectors are reduced by means of POD and assembled in sequences, that are fed to a LSTM autoencoder structure in order to obtain a low-dimensional representation $\tilde{\mathbf{u}}_n(t_i, \boldsymbol{\mu}, \boldsymbol{\theta}_{enc})$ of the entire sequence. A (deep) feedforward neural network is used to infer that low-dimensional representation starting from the initial time t_i and the parameters vector $\boldsymbol{\mu}$ and a LSTM decoder structure allows for the prediction of the sequence of ROM solutions.

The working scheme of the μ-POD-LSTM-ROM method can be divided in the following blocks:

- POD is performed on the snapshot matrix $\boldsymbol{S} \in \mathbb{R}^{N_h \times N_{train} N_t}$ (being N_h the finite discretization dimension of the numerical solver used to produce the snapshots, N_{train} the number of input parameters instances used for building the training set and N_t the number of temporal step used for the time discretization). In the test cases presented, POD was executed in its randomized fashion [34], as in many high-dimensional cases the costs of computing the SVD of the snapshot matrix could become unfeasible [48]. This determines a first dimensionality reduction aimed at making the snapshots dimension suitable for feeding the subsequent neural network part. The projection is then performed as
$$\mathbf{u}_N(t; \boldsymbol{\mu}) = \mathbf{V}_N^T \mathbf{u}_h(t; \boldsymbol{\mu}), \tag{3.1}$$
being $\mathbf{V}_N$ the POD projection matrix.
- Once dimensionally reduced by means of (r)POD, snapshots are sequentially stacked in matrices of the form $\mathbf{V}_N^T \mathbf{u}_h([t_i \dots t_{i+K-1}]; \boldsymbol{\mu}) \in \mathbb{R}^{K \times N}$ where $\mathbf{u}_N([t_i \dots t_{i+K-1}]; \boldsymbol{\mu}) = [\mathbf{u}_N(t_i; \boldsymbol{\mu}), \dots, \mathbf{u}_N(t_{i+K-1}; \boldsymbol{\mu})]$. Such matrices are then grouped in (mini) batches tensors of dimension $\dim_{batch} \times K \times N$ to enable the training.

[*]Note that $t_i \in (0, T - K\Delta t)$ denotes a generic starting time–not necessarily the one of the solution approximation–and that the notation $[t_i \dots t_{i+K-1}]$ (with $t_{i+j} = t_i + j\Delta t$) when used in place of a specific time t indicates the stacking of K subsequent vectors referring to the reported times.

- The sequences of size $K \times N$ from the (mini) batches tensor are then fed to a LSTM encoder structure. The encoder then takes sequentially the $\mathbf{V}_N^T \mathbf{u}_h(t;\boldsymbol{\mu})$ as input and modifies its internal state coherently with the evolution of the vectors it receives. The output of the encoder is neglected as it would be of no utility in this context. In particular, the low-dimensional hidden state representation of the LSTM encoder state is built according to the following function:

$$\tilde{\mathbf{u}}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc}) = \boldsymbol{\lambda}_n^{enc}(\mathbf{u}_N([t_i \dots t_{i+K-1}];\boldsymbol{\mu}),\boldsymbol{\theta}_{enc}), \tag{3.2}$$

  where $\tilde{\mathbf{u}}_n(\cdot;\cdot,\boldsymbol{\theta}_{enc}) : [0,T) \times \mathbb{R}^{n_\mu} \to \mathbb{R}^n$. In the end, the information coming from a sequence of inputs is reduced into a lower dimensional manifold of dimension $n < N \ll N_h$.
- The reconstruction of the low-dimensional hidden state representation of the LSTM encoder, embedding a kind of characterization of the previous time-step data, is performed by a suitable (deep) feedforward neural network consisting in multiple layers of linear transformations and subsequent nonlinear activation functions. The relation learned by this network is

$$\mathbf{u}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN}) = \boldsymbol{\phi}_n^{FFNN}(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN}), \tag{3.3}$$

  with $\mathbf{u}_n(\cdot;\cdot,\boldsymbol{\theta}_{FFNN}) : [0,T) \times \mathbb{R}^{n_\mu} \to \mathbb{R}^n$. Note that just the initial time t_i, along with the current parameter instance, is passed as input to this feedforward neural network but the hidden representation it infers contains the information coming from an entire sequence of K solutions. This low-dimensional representation is then crucial for the compression of information that allows this architecture to work with sequences, as it provides a convenient way to infer the evolution of the solution by considering a single vector as regression target.
- The reduced nonlinear trial manifold $\tilde{\mathcal{S}}_N^n$ is modeled using a LSTM decoder that takes as input the approximated hidden representation coming from the feedforward neural network. In particular, the reduced nonlinear trial manifold can be defined as

$$\begin{aligned} \tilde{\mathcal{S}}_N^n = &\{\boldsymbol{\lambda}_N^{dec}(\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN});\boldsymbol{\theta}_{dec}) \mid \\ &\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN}) \in \mathbb{R}^n,\ t \in [0,T) \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^N, \end{aligned} \tag{3.4}$$

  where $\boldsymbol{\lambda}_N^{dec}(\cdot;\boldsymbol{\theta}_{dec}) : \mathbb{R}^n \to \mathbb{R}^N$, and it can be obtained through a μt-POD-LSTM-ROM. Nevertheless, the novel LSTM cell implementation is able to provide a more meaningful approximated solution manifold to this scope, $\tilde{\mathcal{S}}_{N,K}^n$, that allows for each input tuple $(\boldsymbol{\mu},t)$ the reconstruction of the entire sequence $\tilde{\mathbf{u}}_N([t,\dots,t+(K-1)\Delta t];\boldsymbol{\mu}) \approx \mathbf{u}_N([t,\dots,t+(K-1)\Delta t];\boldsymbol{\mu})$:

$$\begin{aligned} \tilde{\mathcal{S}}_{N,K}^n = &\{\boldsymbol{\lambda}_N^{dec}(\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN});\boldsymbol{\theta}_{dec}) \mid \\ &\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN}) \in \mathbb{R}^n,\ t \in [0,T-K\Delta t) \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N\times K}. \end{aligned} \tag{3.5}$$

  In this context, $\boldsymbol{\lambda}_N^{dec}(\cdot;\boldsymbol{\theta}_{dec}) : \mathbb{R}^n \to \mathbb{R}^{N\times K}$ is a suitable LSTM decoder function, taking as input the hidden state of a LSTM encoder and reconstructing the solution starting from it.
- Once an output sequence has been produced by the decoder function $\boldsymbol{\lambda}_N^{dec}(\cdot;\boldsymbol{\theta}_{dec})$ in (3.5),

$$\tilde{\mathbf{u}}_N([t_i \dots t_{i+K-1}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{dec}) = \boldsymbol{\lambda}_N^{dec}(\mathbf{u}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN});\boldsymbol{\theta}_{dec}), \tag{3.6}$$

  each of its components is expanded from dimension N to dimension N_h by means of the POD basis found before:

$$\tilde{\mathbf{u}}_h([t_i \dots t_{i+K-1}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{dec}) = \mathbf{V}_N \tilde{\mathbf{u}}_N([t_i \dots t_{i+K-1}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{dec}); \tag{3.7}$$

  finally, a stack of N_h full dimensional time sequential solutions is found.

In the notation used above, parameters vectors $\boldsymbol{\theta}_{enc}$, $\boldsymbol{\theta}_{FFNN}$ and $\boldsymbol{\theta}_{dec}$ contain the trainable parameters of the networks. Their hyperparameters (such as, e.g., the number of stacked LSTM cells or their possible bidirectionality or the depth of the deep FFNN) should be considered as well in a separate optimization process.

Remark 1. *The dimensionality of each POD-reduced snapshot composing the input sequence can be, optionally, further reduced by means of a time distributed feedforward neural network before passing through the LSTM autoencoder. This network applies to each* $\mathbf{u}_N(t;\boldsymbol{\mu}) = \mathbf{V}_N^T\mathbf{u}_h(t;\boldsymbol{\mu})$ *in the sequence* $\mathbf{u}_N([t_i \dots t_{i+K-1}];\boldsymbol{\mu})$ *the same nonlinear transformation that reduces its dimensions from N to* N_{red}*. For simplicity, in this work we will assume* $N_{red} = N$*. Furthermore, the low-dimensional representation provided by the autoencoder can be (optionally) compressed by means of a FFNN.*

Remark 2. *Note that* N_h *and N represent the discretization dimensions. In case of vectorial PDE problems, for each time at each spatial discretization point we associate a vector in* $\mathbb{R}^{n_{ch}}$*. In this case the dimensionality of the FOM solution increases from* N_h *to* $N_h n_{ch}$ *and the one of the POD reduced solution increases from N to* $N n_{ch}$*. The structure just described, though, is still valid also in this case provided that the involved dimensions are suitably modified.*

The offline training stage consists of the solution of an optimization problem in which a loss function expressed as a function in the variable $\boldsymbol{\theta} = (\boldsymbol{\theta}_{enc}, \boldsymbol{\theta}_{FFNN}, \boldsymbol{\theta}_{dec})$ should be minimized. In particular, for the training of the μ-POD-LSTM-ROM, the snapshot matrix $\mathbf{S} \in \mathbb{R}^{N_h \times N_{train} N_t}$ (with N_{train} being the number of unique instances drawn from the parameters' space and N_t is the number of timesteps chosen for the time discretization of the interval (0, T)) is compressed by means of POD as explained before to become $\mathbf{S}_{POD} \in \mathbb{R}^{N \times N_{train} N_t}$. Then, sequences from this matrix are extracted to form the so called *base tensor* $\mathbf{T} \in \mathbb{R}^{N_{train}(N_t-K) \times N \times K}$, to be fed to the network,

$$\mathbf{T}(i, j, k) = (\mathbf{u}_N(t_{\alpha_i} + k\Delta t, \boldsymbol{\mu}_{\beta_i}))_j,$$

with $\alpha_i = i \text{mod}(N_t - K)$ and $\beta_i = \frac{i-\alpha_i}{N_t-K}$ and $(\cdot)_j$ denoting the extraction of the j^{th} component from a vector. The minimization problem can therefore be formulated in this case as

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{N_{train}(N_t - K)} \sum_{i=1}^{N_{train}} \sum_{k=1}^{N_t-K} \mathcal{L}(t_k, \boldsymbol{\mu}_i; \boldsymbol{\theta}), \tag{3.8}$$

where the loss function is defined by

$$\mathcal{L}(t_k, \boldsymbol{\mu}_i; \boldsymbol{\theta}) = \frac{\omega_h}{2} \mathcal{L}_{rec}(t_k, \boldsymbol{\mu}_i; \boldsymbol{\theta}) + \frac{1-\omega_h}{2} \mathcal{L}_{int}(t_k, \boldsymbol{\mu}_i; \boldsymbol{\theta}), \tag{3.9}$$

with

$$\mathcal{L}_{rec}(t_k, \boldsymbol{\mu}_i; \boldsymbol{\theta}) = ||\mathbf{T}(i, :, k) - \tilde{\mathbf{u}}_N(t_{\alpha_i} + k\Delta t; \boldsymbol{\mu}_{\beta_i}, \boldsymbol{\theta}_{FFNN}, \boldsymbol{\theta}_{dec})||^2$$

and

$$\mathcal{L}_{int}(t_k, \boldsymbol{\mu}_i; \boldsymbol{\theta}) = ||\tilde{\mathbf{u}}_n(t_{\alpha_i} + k\Delta t; \boldsymbol{\mu}_{\beta_i}, \boldsymbol{\theta}_{enc}) - \mathbf{u}_n(t_{\alpha_i} + k\Delta t; \boldsymbol{\mu}_{\beta_i}, \boldsymbol{\theta}_{FFNN})||^2,$$

with $|| \cdot ||$ intended as the Euclidean norm $|| \cdot ||_2$. The loss function (3.9) penalizes the reconstruction error from the LSTM autoencoder through $\mathcal{L}_{rec}$ and the difference between the low-dimensional hidden representation learned by the encoder and the prediction from the feedforward neural network fed with the problem parameters through $\mathcal{L}_{int}$. The coefficient $\omega_h \in [0,1]$ regulates the importance of the two components of the loss function.

During the online stage (at testing), just the feedforward part $\boldsymbol{\phi}_N^{FFNN}(\cdot;\cdot,\boldsymbol{\theta}_{FFNN})$ and the decoder $\boldsymbol{\lambda}_N^{dec}(\cdot;\boldsymbol{\theta}_{dec})$ are used. The encoder part is added at training time in order to help the network learning the correct hidden representation of the sequences in a data-driven fashion.

4. *t*-POD-LSTM-ROM

The *t*-POD-LSTM-ROM is the second component of the *μt*-POD-LSTM-ROM framework, providing it with time extrapolation capabilities. As the name suggests, it works on time series forecasting by solving iteratively a *sequence to sequence* (also referred to as *seq2seq*) learning problem, that can be defined (see, e.g., [49]) as the forecasting of a certain number (k) of steps ahead in a time series $y(t)$. Therefore, its solution provides a model (the *predictor*), that takes as input p time steps in the past and returns as output the forecasted k steps ahead in the future. The predictor is a function taking as input a sequence, and returning a sequence as output as well

$$[y(t-p+1), y(t-p+2), \dots, y(t)] \mapsto [\hat{y}(t+1), \hat{y}(t+2), \dots, \hat{y}(t+k)]. \tag{4.1}$$

Traditional machine learning models, such as simple regression, support vector regression, ARIMA and feedforward neural networks have been used to tackle the problem [50]. Hidden Markov models or fuzzy logic based models have also proven to be somehow effective in the field [51, 52]. Recently though, artificial neural networks featuring recurrence mechanisms such as simple recurrent neural networks or LSTM cells have become the standard for time series prediction when having large amount of data available for the training [49, 50, 53, 54]. The idea of a windowed auto-regressive prediction has been exploited in neural ODEs [55, 56], where deterministic numerical solvers consider also statistically learned residuals in order to perform the PDE integration in time. This method is very effective in time extrapolation capabilities, but still considers numerical integration of high dimensional systems. A similar approach exploited in the field of ROMs has been considered in [28], where a correction parameter was included in the numerical time integration process. Since the use of a numerical integrator can negatively impact the time performances of the method, we decided to rely on a time series forecasting problem as described before. In particular, in [57] a time series approach for PDE problems using LSTMs proved to be effective in forecasting reduced barotropic climate models and showing interesting time performances, however neglecting in that case the parametric nature of the problem. Note that considering architectures based on the *seq2seq* paradigm shows close similarities with the behavior of numerical solvers, as they build predictions for future times based on the past, mimicking time marching numerical schemes.

In this work, we introduce the *t*-POD-LSTM-ROM architecture, aimed at solving the *seq2seq* problem in the context of ROMs for parameterized dynamical systems. In particular, the novel framework allows to extend in the temporal dimension the ROM solution provided by the *μ*-POD-LSTM-ROM framework introduced before, by considering the *seq2seq* problem on the reduced order vectors. The training of this additional neural network does not require an increased number of snapshots, keeping the temporal cost of the offline phase relatively low. Furthermore, the possibility to extend the temporal domain of definition of the solution could in principle allow for the training using full order snapshots defined on a shorter period of time, thus requiring less computational expenses when generating them.

The architecture of the t-POD-LSTM-ROM, summarized in Figure 3, consists of the components listed below:

- The same POD-reduced sequences $\mathbf{V}_N^T\mathbf{u}_h([t_{i-p+1}\dots t_i];\boldsymbol{\mu})$ created to feed the μ-POD-LSTM-ROM framework, and formed by the $p-1$ snapshots preceding the one at time t_i and the one at time t_i itself, are collected in pairs with $\mathbf{V}_N^T\mathbf{u}_h([t_{i+1}\dots t_{i+k}];\boldsymbol{\mu})$, the sequence formed by the k following reduced snapshots to be used in the training phase.
- A LSTM encoder is then applied to provide a low-dimensional representation of the data extracted by the past time steps sequence. In particular, the encoder acts according to

$$\mathbf{h}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc}) = \boldsymbol{\Lambda}_n^{enc}(\mathbf{V}_N^T\mathbf{u}_h([t_{i-p+1}\dots t_i];\boldsymbol{\mu});\boldsymbol{\theta}_{enc}). \tag{4.2}$$

  The low-dimensional vector containing an embedded representation of the input which is used to build the sequence of future solutions by means of a LSTM decoder.
- A feedforward neural network is then used to provide information on the parameter instance considered to the architecture. First, some dense layers are applied in order to expand the information contained in the parameters' vector $\boldsymbol{\mu}$ and then the result of this operation is concatenated to the low-dimensional representation as $\mathbf{h}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc}) \oplus \boldsymbol{\phi}^1(\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN1})$[†] Differently from μ-POD-LSTM-ROM, the output of the LSTM encoder and the one of the feedforward neural network are not matched but rather concatanated; indeed, in t-POD-LSTM-ROM the two outputs represent different quantities. This concatenation is then fed to another feedforward neural network aimed at merging the information coming from the past snapshots and the one coming from the parameters in order to build a more accurate low-dimensional representation of the data, of the form

$$\mathbf{h'}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc},\boldsymbol{\theta}_{FFNN}) = \boldsymbol{\phi}^2(\boldsymbol{\mu},[\mathbf{h}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc})\oplus\boldsymbol{\phi}^1(\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN1})];\boldsymbol{\theta}_{FFNN2}), \tag{4.3}$$

  with $\boldsymbol{\theta}_{FFNN} = (\boldsymbol{\theta}_{FFNN1},\boldsymbol{\theta}_{FFNN2})$, being $\boldsymbol{\theta}_{FFNN1}$ and $\boldsymbol{\theta}_{FFNN2}$ the vectors of parameters of the two feedforward neural network parts, namely $\boldsymbol{\phi}^1$ and $\boldsymbol{\phi}^2$.
- A LSTM decoder then takes the informed low-dimensional representation $\mathbf{h'}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc},\boldsymbol{\theta}_{FFNN})$ as input, and extracts the forecast reduced future sequence according to

$$\tilde{\mathbf{u}}_N([t_{i+1}\dots t_{i+k}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{enc},\boldsymbol{\theta}_{dec}) = \boldsymbol{\Lambda}_n^{dec}(\mathbf{h'}_n(t_i;\boldsymbol{\mu},\boldsymbol{\theta}_{enc},\boldsymbol{\theta}_{FFNN})). \tag{4.4}$$

  From this output, the sequence of full order predicted solutions is then reconstructed by means of the POD basis. In contrast with the μ-POD-LSTM-ROM technique considered before (and the POD-DL-ROM one), in this case the architecture of the autoencoder–and therefore that of the entire network–remains the same during both the training and the testing stages.

[†]We define in this context $\oplus$ with the concatenation between two vectors by appending the second after the first. By defining $\mathbf{a}\in\mathbb{R}^{n_1}$ and $\mathbf{b}\in\mathbb{R}^{n_2}$, then $\mathbf{a}\oplus\mathbf{b} = [\mathbf{a},\mathbf{b}] = \mathbf{c}\in\mathbb{R}^{n_1+n_2}$.

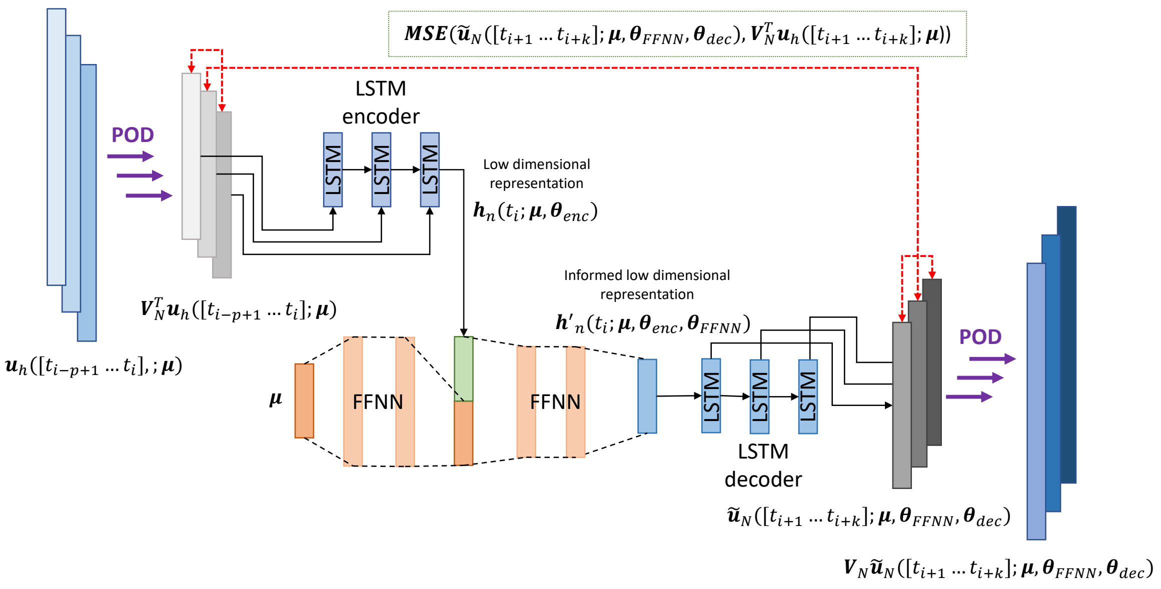
$MSE(\tilde{u}_N([t_{i+1} \dots t_{i+k}]; \mu, \theta_{FFNN}, \theta_{dec}), V_N^T u_h([t_{i+1} \dots t_{i+k}]; \mu))$
LSTM encoder
POD
LSTM
Low dimensional representation
$h_n(t_i; \mu, \theta_{enc})$
$V_N^T u_h([t_{i-p+1} \dots t_i]; \mu)$
$u_h([t_{i-p+1} \dots t_i], ; \mu)$
Informed low dimensional representation
$h'_n(t_i; \mu, \theta_{enc}, \theta_{FFNN})$
μ
FFNN
FFNN
LSTM decoder
$\tilde{u}_N([t_{i+1} \dots t_{i+k}]; \mu, \theta_{FFNN}, \theta_{dec})$
$V_N \tilde{u}_N([t_{i+1} \dots t_{i+k}]; \mu, \theta_{FFNN}, \theta_{dec})$

Figure 3. The *t*-POD-LSTM-ROM architecture. After an initial POD-based dimensionality reduction, the sequence of reduced vectors passes through a LSTM encoder. The low-dimensional representation $\mathbf{h}_n(t_i; \boldsymbol{\mu}, \boldsymbol{\theta}_{enc})$ thus obtained is then enriched with the information coming from the problem's parameters and starting from it a LSTM decoder finally extracts the *k*-steps forecast sequence.

Also in this case, the parameters vectors $\boldsymbol{\theta}_{enc}$, $\boldsymbol{\theta}_{FFNN}$ and $\boldsymbol{\theta}_{dec}$ group the trainable weights and biases of the networks. The training of the network described above is then performed as an optimization problem in which a loss function expressed in the variable $\boldsymbol{\theta} = (\boldsymbol{\theta}_{enc}, \boldsymbol{\theta}_{FFNN}, \boldsymbol{\theta}_{dec})$ should be minimized. After the definition of the *base tensor* $\mathbf{T} \in \mathbb{R}^{N_{train}(N_t-K)\times N\times K}$, as described when introducing a μ-POD-LSTM-ROM, with $K = k + p$, we divide it in two parts: the *previous steps* tensor $\mathbf{P} \in \mathbb{R}^{N_{train}(N_t-K)\times N\times p}$ and the *horizon* tensor $\mathbf{H} \in \mathbb{R}^{N_{train}(N_t-K)\times N\times k}$. In particular, we define

$$\mathbf{P} = \mathbf{T}(:, :, 1 : p) \text{ and } \mathbf{H} = \mathbf{T}(:, :, (p + 1) : K), \tag{4.5}$$

so that the tensor $\mathbf{P}$ represents the previous time steps to be used for the prediction and $\mathbf{H}$ contains the target sequences to forecast. In the end, the minimization problem solved during training can be defined as

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{N_{train}(N_t - K)} \sum_{i=1}^{N_{train}} \sum_{j=1}^{N_t-K} \mathcal{L}(t_j, \boldsymbol{\mu}_i; \boldsymbol{\theta}), \tag{4.6}$$

where we define

$$\mathcal{L}(t_j, \boldsymbol{\mu}_i; \boldsymbol{\theta}) = \text{MSE}\left[\tilde{\mathbf{u}}_N([t_{j+1} \dots t_{j+k}]; \boldsymbol{\mu}_i, \boldsymbol{\theta}_{FFNN}, \boldsymbol{\theta}_{enc}, \boldsymbol{\theta}_{dec}),\ \mathbf{V}_N^T \mathbf{u}_h([t_{j+1} \dots t_{j+k}]; \boldsymbol{\mu}_i)\right] \tag{4.7}$$

with

$$\begin{aligned}\mathrm{MSE}\left[\tilde{\mathbf{u}}_N([t_{j+1}\dots t_{j+k}];\boldsymbol{\mu},\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{enc},\boldsymbol{\theta}_{dec}),\ \mathbf{V}_N^T\mathbf{u}_h([t_{j+1}\dots t_{j+k}];\boldsymbol{\mu})\right]\\ = \frac{1}{Nk}\sum_{l=1}^{N}\sum_{p=1}^{k}\left(\left(\tilde{\mathbf{u}}_N(t_{j+p};\boldsymbol{\mu}_i,\boldsymbol{\theta}_{FFNN},\boldsymbol{\theta}_{enc},\boldsymbol{\theta}_{dec})\right)_l - \left(\mathbf{V}_N^T\mathbf{u}_h(t_{j+p};\boldsymbol{\mu}_i)\right)_l\right)^2.\end{aligned}$$

The loss function (4.7) therefore penalizes prediction errors and maximizes the accuracy in prediction.

5. Results

In this section we present a set of numerical results obtained on three different test cases, related with *(i)* a 3 species Lotka-Volterra equations (Section 5.1), *(ii)* unsteady advection-diffusion-reaction equation (Section 5.3), *(iii)* incompressible Navier-Stokes equations (Section 5.5). To assess the accuracy of the numerical results, we consider the same two error indicators defined in [33], namely:

- the error indicator $\epsilon_{rel} \in \mathbb{R}$ defined as

$$\epsilon_{rel}(\mathbf{u}_h,\tilde{\mathbf{u}}_h) = \frac{1}{N_{test}}\sum_{i=1}^{N_{test}}\left(\frac{\sqrt{\sum_{k=1}^{N_t}\|\mathbf{u}_h^k(\boldsymbol{\mu}_{test,i})-\tilde{\mathbf{u}}_h^k(\boldsymbol{\mu}_{test,i})\|^2}}{\sqrt{\sum_{k=1}^{N_t}\|\mathbf{u}_h^k(\boldsymbol{\mu}_{test,i})\|^2}}\right), \tag{5.1}$$

- the relative error $\boldsymbol{\epsilon}_k \in \mathbb{R}^{\sum_i N_h^i}$, for $k = 1,\dots,N_t$, defined as

$$\boldsymbol{\epsilon}_k(\mathbf{u}_h,\tilde{\mathbf{u}}_h) = \frac{|\mathbf{u}_h^k(\boldsymbol{\mu}_{test})-\tilde{\mathbf{u}}_h^k(\boldsymbol{\mu}_{test})|}{\sqrt{\frac{1}{N_t}\sum_{k=1}^{N_t}\|\mathbf{u}_h^k(\boldsymbol{\mu}_{test})\|^2}}. \tag{5.2}$$

Note that $\mathbf{u}_h^k(\boldsymbol{\mu}_{test,i}) = \mathbf{u}_h(t_k;\boldsymbol{\mu}_{test,i})$ and $\tilde{\mathbf{u}}_h^k(\boldsymbol{\mu}_{test,i}) = \tilde{\mathbf{u}}_h(t_k;\boldsymbol{\mu}_{test,i})$. Moreover, the error indicator ϵ_{rel} provides a (scalar) numerical estimation of the accuracy of the method on the entire test set.

To solve the optimization problems, we use the ADAM algorithm [58], which is a stochastic gradient descent method computing an adaptive approximation of the first and second momentum of the gradients of the loss function. In particular, it computes exponentially weighted moving averages of the gradients and of the squared gradients. We set the starting learning rate to $\eta = 10^{-4}$, and perform cross-validation in order to tune the hyperparameters of the neural networks by splitting the data in training and validation sets with a proportion 9:1. Moreover, we implement an early-stopping regularization technique to reduce overfitting [59], stopping the training if the loss does not decrease over a certain amount of epochs. As nonlinear activation function, we employ the ELU function [60]. The parameters, weights, and biases are initialized through He uniform initialization [61].

The μ-POD-LSTM-ROM and the t-POD-LSTM-ROM architectures have been developed using the TensorFlow 2.4 framework [62]. Details on these architectures are provided in each example. Regarding the setting of the optimization algorithm and the way to select the hyperparameters, we refer to [33]. FOM data for test cases (ii) and (iii) have been obtained by the redbKIT v2.2 library [63], implementing the methods described in [2]. All the simulations have been run on an Intel® Core i9 @ 2.40GHz CPU, 16 GB RAM and NVIDIA® GTX1650 video card personal computer. The algorithms through which the training and testing phases of the networks are performed can be found in the Appendix.

5.1. Lotka-Volterra competition model (3 species)

The first test case is the 3 species Lotka-Volterra competition model, selected to provide a proof-of-concept of the method on a simple but aperiodic test case. The goal is the reconstruction of the solution $\mathbf{u} = \mathbf{u}(t;\mu) \in \mathbb{R}^3$ of the following system:

$$\begin{cases} \dfrac{du_1}{dt}(t) = u_1(t)(\mu - 0.1u_1(t) - 0.5u_2(t) - 0.5u_3(t)) & t \in (0,T), \\ \dfrac{du_2}{dt}(t) = u_2(t)(-\mu + 0.5u_1(t) - 0.3u_3(t)) & t \in (0,T), \\ \dfrac{du_3}{dt}(t) = u_3(t)(-\mu + 0.2u_1(t) + 0.5u_2(t)) & t \in (0,T), \\ u_i(0) = 0.5 & \forall i \in \{1,2,3\}. \end{cases} \tag{5.3}$$

Note that due to the low-dimensionality of the problem ($N_h = 3$), in this case the use of POD is not necessary and therefore it is not performed. Nevertheless, the entire deep learning-based architecture is still used, providing a first glance on the performances of the presented framework when considering time extrapolation capabilities. The parameter $\mu \in \mathcal{P} = [1,3]$ models both the reproduction rate of the species 1 (the prey) and the mortality rate of species 2 and 3 (predators), assumed to be equal. The impact of μ on the solution concerns both the amplitude and the frequency of the oscillation of the 3 species' populations. The equations have been discretized by means of an explicit Runge-Kutta (4,5) formula with a time step $\Delta t = 0.1$ over the time interval $(0,T)$, with $T = 9.9$.

The LSTM-ROM framework used to find the solution of the system considers $N_t = 100$ time instances with $N_{train} = 21$. In particular, the selected μ for the training are equally spaced in the interval $\mathcal{P} = [1,3]$ (that is, $\mathcal{P}_{train} = \{1, 1.1, 1.2, \dots, 2.9, 3\}$). The LSTM sequence length used for the training is $K = 20$, the hidden dimension of the LSTM network has been chosen to be $n = 40$ and the loss parameter ω_h introduced in (3.9) has been set equal to $\omega_h = 0.9$. These choices are the result of a random search hyperparameters tuning [64] considering both accuracy and time performances. The number of epochs have been fixed to a maximum of $n_{epochs} = 4000$ with the early stopping criterion intervening after 50 epochs of missed improvement of the loss function over the validation set during optimization.

We present here, for the sake of comparison, also the results obtained with the DL-ROM framework, for which we considered the same training set used to train the μ-LSTM-ROM network with the same maximum number of epochs and early stopping criterion. The low-dimensional manifold representation was $n = 40$ for the μ-LSTM-ROM and $n = 10$ for the DL-ROM. Note that in this case $n > N_h$. This happens in the context of μ-LSTM-ROMs because the low-dimensional representation should contain enough information for the decoder to reconstruct an entire sequence of $N_h = 3-$dimensional vectors of length $K = 20$ resulting in a $N_h \times K = 3 \times 20 = 60$ components output. Also a DL-ROM in this case showed better performances with $n = 10 > n_\mu + 1$, probably due to the extremely low magnitude of N_h. In general though, the proposed LSTM-based framework requires larger dimensional reduced manifolds with respect to a POD-DL-ROM in order to provide the decoder with enough information to reconstruct POD-reduced solution sequences.

Regarding the architectures considered, we chose to rely on a much larger LSTM-ROM architecture (34433 trainable parameters) with respect to the DL-ROM one used (2943 trainable parameters). This unbalance is recurrent in all the test cases presented. A μ-POD-LSTM-ROM requires in fact a larger number of parameters as it needs to encode more information with respect to a POD-DL-ROM. The μ-LSTM-ROM training took 3124 epochs (932 s), while the DL-ROM one took 1441 epochs (274s). Results in terms of time evolution of the 3 species for a representative instance of parameters space ($\mu = 1.95$, equally distant from the extremes of $\mathcal{P}_{test}$) are reported in Figure 4.

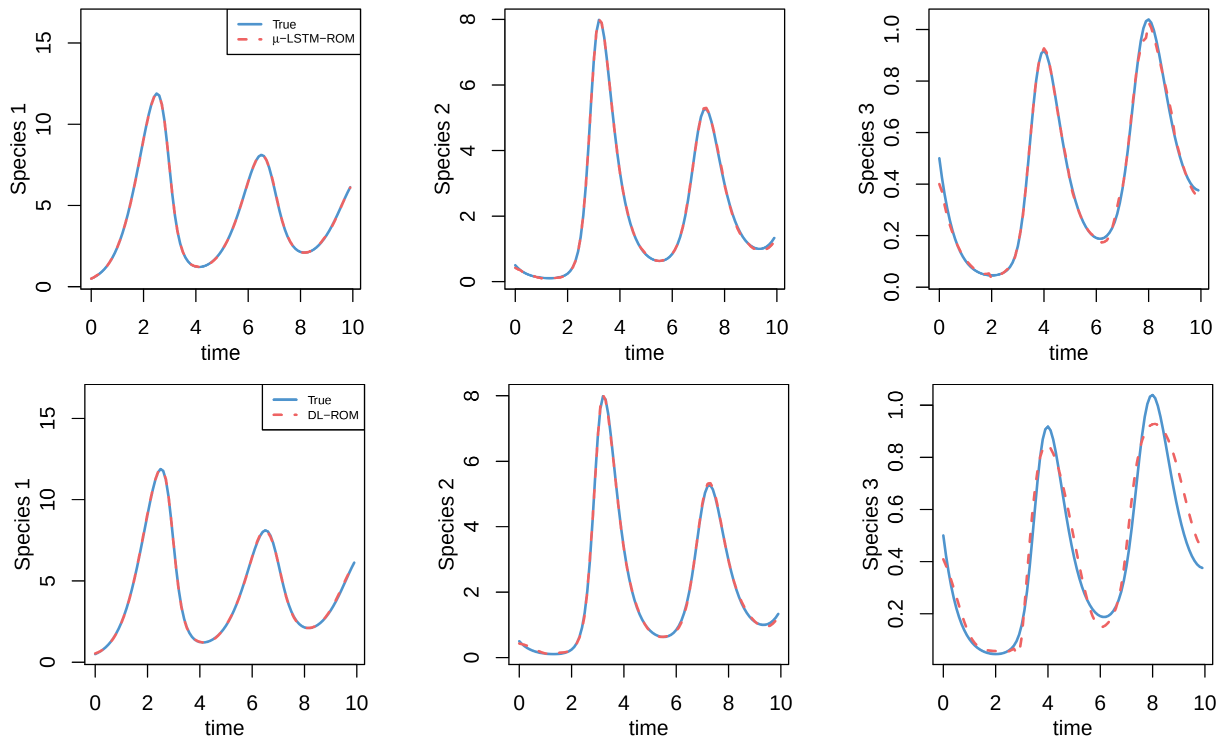

Figure 4. Test case 1–Lotka-Volterra system. Simulated results for $\mu = 1.95$. Top: μ-LSTM-ROM framework, bottom: DL-ROM framework.

The error indicator $\epsilon_{rel}^{LSTM-ROM}$ for the LSTM-ROM case is $5.582 \cdot 10^{-3}$, while for the DL-ROM we find $\epsilon_{rel}^{DL-ROM} = 1.363 \cdot 10^{-2}$. Hence the novel framework provides slightly better results in terms of accuracy in this particular application. Table 1 reports the obtained relative error ϵ_k, together with the 95% bootstrap confidence intervals for the mean relative error $\boldsymbol{\epsilon}_{\mathbf{k}}^{\mathbf{mean}}$.

Table 1. Test case 1–Lotka-Volterra system. Error results and 95% bootstrap confidence intervals in comparison between μ-LSTM-ROM and DL-ROM methods.

	$\boldsymbol{\epsilon}_{\mathbf{k}}^{\mathbf{mean}}$	$\boldsymbol{\epsilon}_{\mathbf{k}}^{\mathbf{max}}$	$C_{0.95}^{LSTM-ROM}$
μ-LSTM-ROM	$4.836 \cdot 10^{-4}$	$6.703 \cdot 10^{-3}$	$[4.660 \cdot 10^{-4}, 5.007 \cdot 10^{-4}]$
DL-ROM	$1.076 \cdot 10^{-3}$	$1.082 \cdot 10^{-2}$	$[1.040 \cdot 10^{-3}, 1.110 \cdot 10^{-3}]$

The loss in accuracy regarding both methods when considering the third species has to be accounted to the smaller scales characterizing it. Furthermore, we need to consider the disparity in the number of parameters among the two neural networks. Both architectures though are extremely simple and the DL-ROM one includes the deepest networks and it is the result of a the composition of a higher number of nonlinear layers; hence these features should partly compensate for its lower parameters' number.

Regarding prediction times, we consider them from a double perspective. We will report the crude time t_{NN} that the neural network structure takes to perform the forward pass and thus produce a prediction of the solution for each instance of the parameters' test space $\mathcal{P}_{test}$, and the total time t_{rec} which considers also the construction of the full order vectors. Note that the μ-POD-LSTM-ROM neural network would require $\frac{1}{K}$ input-output pairs with respect to the POD-DL-ROM one in this phase, because it needs just one prediction every K time steps–it deals with sequences of data and not just with a single time instance–to be able to build the time evolution of the solution. The quantity t_{rec} takes also into account the amount of time required to assemble the actual solutions, but this is strictly code and language dependent. The bottleneck in computations is in our opinion represented by t_{NN}.

For the case at hand we found, after a run of 100 queries on the entire $\mathcal{P}_{test}$, the times reported in Table 2. A 34.6% reduction in t_{NN}^{mean} and a 31.0% reduction in t_{rec}^{mean} can be observed when using a μ-LSTM-ROM instead of a DL-ROM, thus highlighting an increased efficiency of the new method with respect to the old one also when considering the entire solution reconstruction phase. Moreover, the time entailed by the solution of the FOM, by means of an explicit Runge-Kutta (4,5) formula on the interval $(0, T)$, is equal to 1 s.

Table 2. Test case 1–Lotka-Volterra system. Temporal results for the comparison between μ-LSTM-ROM and DL-ROM frameworks.

	$\mathbf{t_{NN}^{min}}$	$\mathbf{t_{NN}^{mean}}$	$\mathbf{t_{NN}^{max}}$	$\mathbf{t_{rec}^{min}}$	$\mathbf{t_{rec}^{mean}}$	$\mathbf{t_{rec}^{max}}$
μ-LSTM-ROM	0.0364 s	0.0415 s	0.0598 s	0.0381 s	0.0446 s	0.3281 s
DL-ROM	0.0583 s	0.0635 s	0.0813 s	0.0587 s	0.0646 s	0.0802 s

5.2. *Lotka-Volterra time extrapolation*

We then tested the time extrapolation capabilities of the method both in the short and the long term. In particular, we present the results of the μt-LSTM-ROM framework for the same training and test sets ($\mathcal{P}_{train}$ and $\mathcal{P}_{test}$) just considered. In this case, though, the training snapshots have been acquired from the time interval $(0, T)$, with $T = 9.9$ and $\Delta t = 0.1$, while the testing ones are taken in $[T_{in}, T_{fin}]$, where $T_{in} = 5$ and $T_{fin} = 14.9$. Training parameters are the same as in the interpolation case. We therefore consider an extrapolation window of length $T_{ext} = 5$, half of the training time interval length.

The t-LSTM-ROM architecture used in addition to the LSTM-ROM one already described considers $p = 10$ previous time steps in order to make inference on a $k = 10$ time steps horizon. Its training, performed without accounting for early stopping, took 1000 epochs (355 s). The plots in Figure 5 show the extrapolation performances of the method when applied to this test case.

For the problem considered the relative error ϵ_k obtained in the aforementioned time extrapolation context is summarized in Table 3. Extrapolation performances of the presented framework are very satisfying also considering long extrapolation windows, as shown for $\mu = 2.45$ in Figure 6. In particular, the temporal evolution of the problem on a time window which is 10 times larger than the training domain is predicted with remarkable accuracy as well as the equilibrium asymptotes.

Table 3. Test case 1–Lotka-Volterra system. Relative error indicators for the μt-POD-LSTM-ROM framework applied to the aperiodic Lotka-Volterra system.

$\epsilon_{\mathbf{k}}^{\mathbf{mean}}$	$\epsilon_{\mathbf{k}}^{\mathbf{max}}$
$1.341 \cdot 10^{-3}$	$3.241 \cdot 10^{-2}$

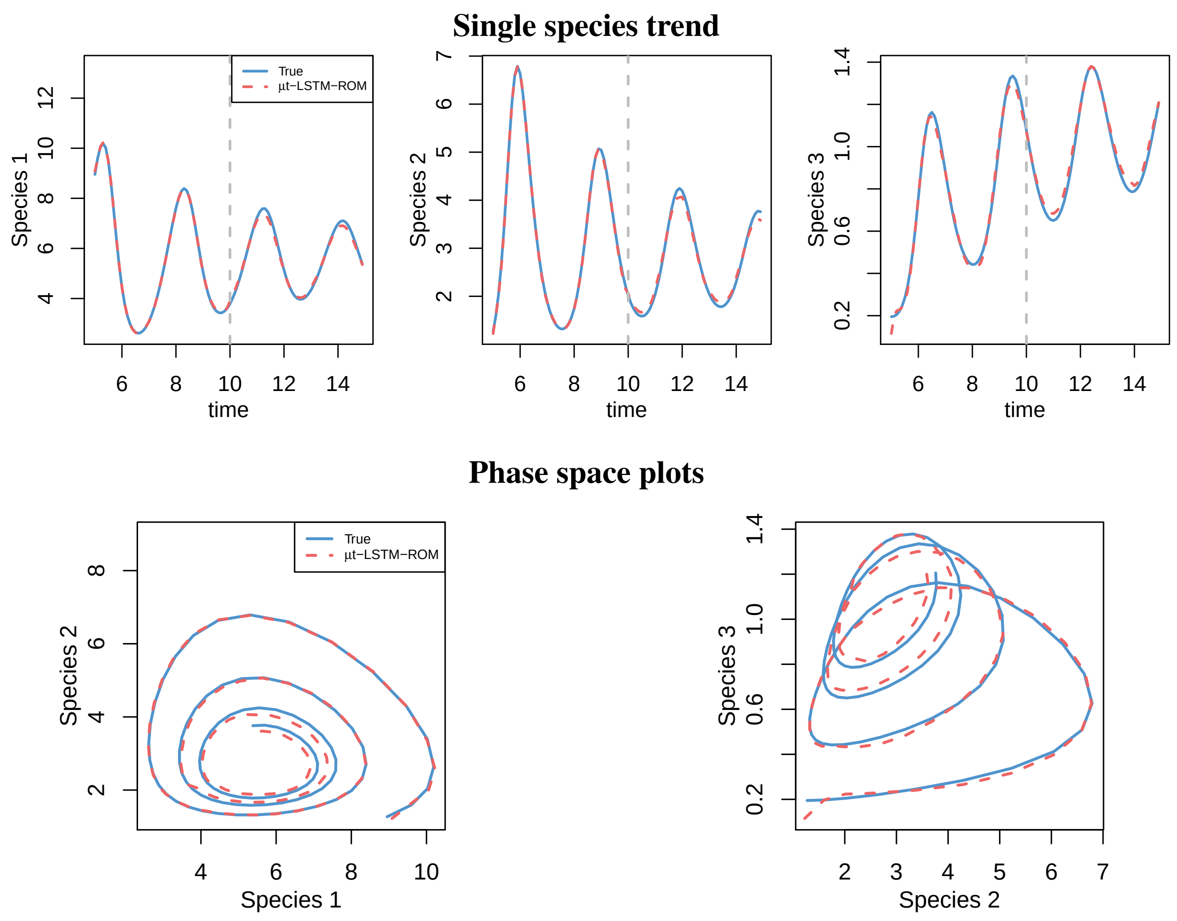

Figure 5. Test case 1–Lotka-Volterra system. Simulation results considering the aperiodic Lotka-Volterra model for $\mu = 1.25$, μt-LSTM-ROM framework. Grey dotted line indicates the starting extrapolation time.

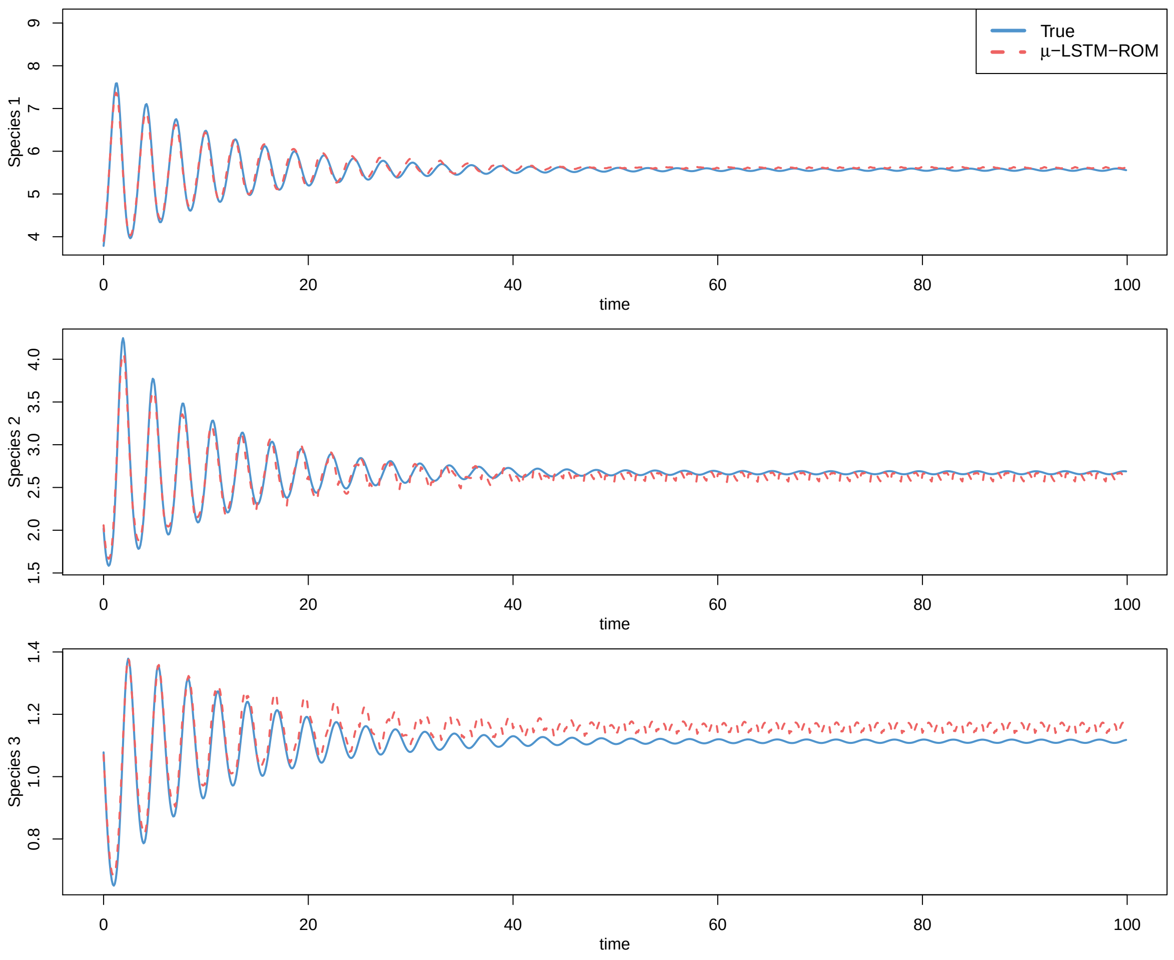

Figure 6. Test case 1–Lotka-Volterra system. Long term ($T_{ext} = 100$) time extrapolation for the evolution of the three species in the aperiodic Lotka-Volterra case.

5.3. Unsteady advection-diffusion-reaction equation

We now consider the case of a parameterized unsteady advection-diffusion-reaction (ADR) problem. In particular, our goal in this case is to approximate the solution $u = u(\mathbf{x}, t; \boldsymbol{\mu})$ of a linear parabolic PDE initial value problem of the following form:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \operatorname{div}(\mu_1 \nabla u) + \mathbf{b}(t) \cdot \nabla u + cu = f(\mu_2, \mu_3), & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \mu_1 \nabla u \cdot \mathbf{n} = 0, & (\mathbf{x}, t) \in \partial\Omega \times (0, T), \\ u(0) = 0 & \mathbf{x} \in \Omega, \end{cases} \tag{5.4}$$

in the two-dimensional domain $\Omega = (0, 1)^2$, where

$$f(\mathbf{x}; \mu_2, \mu_3) = 10 \exp(-((x - \mu_3)^2 + (y - \mu_4)^2)/0.07^2) \qquad \text{and} \qquad \mathbf{b}(t) = [\cos(t), \sin(t)]^T.$$

Regarding the parameters, we took $n_\mu = 3$ parameters belonging to $\mathcal{P} = [0.002, 0.005] \times [0.4, 0.6]^2$. A similar framework has been considered in [33]. In particular we discretized the training parameters space $\mathcal{P}_{train}$ with $\mu_1 \in \{0.002, 0.003, 0.004, 0.005\}$ and $(\mu_2, \mu_3) \in \{0.40, 0.45, 0.50, 0.55, 0.60\}^2$, for a total $N_{train} = 100$ different instances in $\mathcal{P}_{train}$. Parameters μ_2 and μ_3 influence the location of the distributed source in the spatial domain, while the dependence on μ_1 impacts on the relative importance between the advection and the diffusion terms.

Note that the dependence of the solution on μ_2 and μ_3 is nonlinear and therefore the problem is nonaffinely parametrized. This would have required the extensive use of hyper-reduction techniques such as, e.g., the discrete empirical interpolation method (DEIM) [65] in order to properly address the construction of a projection-based ROM exploiting, e.g., the reduced basis method, thus reducing the performance of this latter. The FOM snapshots have been obtained by means of a spatial discretization obtained with linear ($\mathbb{P}_1$) finite elements considering $N_h = 10657$ degrees of freedom (DOFs) and a time discretization relying on a Backward Differentiation Formula (BDF) of order 2. The time step used for the time discretization is $\Delta t = 2\pi/20$ on $(0, T)$ with $T = 10\pi$.

The μ-POD-LSTM-ROM network used to tackle this problem was trained on $N_t = 100$ time steps for each instance of the parameters space. The dimension of the hidden representation of the LSTM autoencoder has been fixed to $n = 100$. In particular, the network is composed by an initial dense part aimed at reducing the POD dimensions to properly feed an encoder composed by two stacked LSTM cells and then a further dense part is traversed in order to reach the reduced dimension state. The decoder is then symmetrically composed by a dense part, two stacked LSTM cells and another dense network to expand the output up to the required N dimension, for a total number of trainable parameters equals to $|\boldsymbol{\theta}| = 402154$.

The POD dimension was set to be $N = 64$ and the randomized version of the POD has been performed in order to reduce the computational effort required by this stage. After POD, the reduced order snapshots have been scaled in the $[0, 1]$ range. This choice has been taken because of the low magnitude of the solution, ranging in $[0, 0.1]$. In such cases, a *MinMax normalization*, defined as

$$x' = \frac{x - min(x)}{max(x) - min(x)} \in [0, 1]$$

has proven to be useful when applied to the reduced order vectors for maximizing neural network performances [66].

Also in this case, we consider the POD-DL-ROM framework for the comparison with a μ-POD-LSTM-ROM. In particular, the POD-DL-ROM architecture used in this context is the one considered in [33] and it is based on a convolutional architecture aimed at the dimensionality reduction of the input. Therefore, the architecture relies on an initial convolution followed by a feed forward neural network for the encoder, and then symmetrically another feed forward neural network and some deconvolution layers in order to reconstruct the spatial dependence of the solution, for a total number of trainable parameters of $|\boldsymbol{\theta}| = 269207$. The training of the μ-POD-LSTM-ROM took 1243 epochs (2145 s), while the one of the POD-DL-ROM took 591 epochs (1121 s). The obtained POD-DL-ROM and the μ-POD-LSTM-ROM results for an instance of the testing parameters space ($\boldsymbol{\mu} = (0.0025, 0.4250, 0.4250)$) are reported in Figure 7.

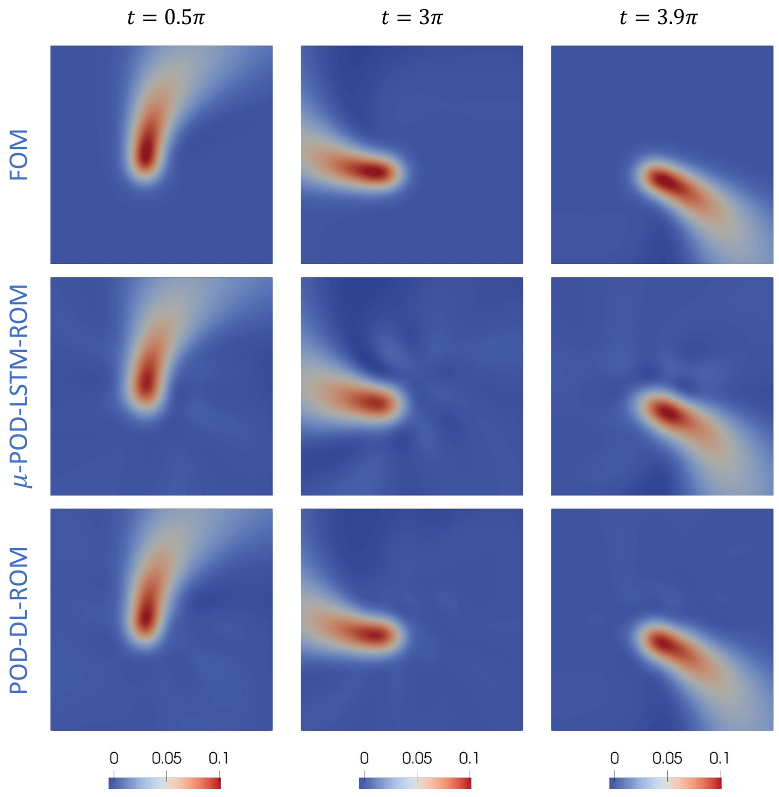

Figure 7. Test case 2–ADR equation. Simulation results for $\boldsymbol{\mu} = (0.0025, 0.4250, 0.4250)$. Top: FOM results, center: μ-POD-LSTM-ROM results, bottom: POD-DL-ROM results.

The error indicator $\epsilon_{rel}^{POD-LSTM-ROM}$ for the LSTM-ROM case is $8.123 \cdot 10^{-2}$, while for the POD-DL-ROM case we find $\epsilon_{rel}^{POD-DL-ROM} = 4.290 \cdot 10^{-2}$, regarding these latter, the obtained results are compatible with those observed in [33]. Regarding the relative error $\boldsymbol{\epsilon}_k$, we found the results reported in Table 4. The 95% bootstrap confidence intervals for the mean relative error ϵ_k^{mean} are for the μ-POD-LSTM-ROM case $C_{0.95}^{POD-LSTM-ROM} = [2.523 \cdot 10^{-4}, 2.571 \cdot 10^{-4}]$ and $C_{0.95}^{POD-DL-ROM} = [1.303 \cdot 10^{-4}, 1.316 \cdot 10^{-4}]$ in the POD-DL-ROM case, showing a better overall accuracy performance of a POD-DL-ROM. Moreover, we have observed in our experiments that the performances of the POD-DL-ROM improve significantly when scaling the input, while the μ-POD-LSTM-ROM seems to be more robust to non-scaled data.

Table 5 summarizes the testing time performances of the μ-POD-LSTM-ROM in comparison with a POD-DL-ROM. Also in this case we report the results in terms both of the neural network involved time and reconstruction time (see the Lotka-Volterra results for details). We obtained a 52.0% decrease in t_{NN}^{mean} and a 14.5% decrease in t_{rec}^{mean} by using a μ-POD-LSTM-ROM over POD-DL-ROM. Also in this case, we report the time required by the solution of the FOM for a single parameter-instance on the interval $(0, T)$ given by 51 s.

In conclusion, μ-POD-LSTM-ROM results to be more efficient than the POD-DL-ROM technique by preserving, at the same time, high levels of accuracy.

Table 4. Test case 2–ADR equation. Error indicators in comparison between μ-POD-LSTM-ROM and POD-DL-ROM frameworks.

	$\boldsymbol{\epsilon}_{\mathbf{k}}^{\mathbf{mean}}$	$\boldsymbol{\epsilon}_{\mathbf{k}}^{\mathbf{max}}$
μ-POD-LSTM-ROM	$2.547 \cdot 10^{-4}$	$3.864 \cdot 10^{-3}$
POD-DL-ROM	$1.309 \cdot 10^{-4}$	$2.103 \cdot 10^{-3}$

Table 5. Test case 2–ADR equation. Temporal results for the comparison between μ-POD-LSTM-ROM and POD-DL-ROM.

	$\mathbf{t}_{\mathbf{NN}}^{\mathbf{min}}$	$\mathbf{t}_{\mathbf{NN}}^{\mathbf{mean}}$	$\mathbf{t}_{\mathbf{NN}}^{\mathbf{max}}$	$\mathbf{t}_{\mathbf{rec}}^{\mathbf{min}}$	$\mathbf{t}_{\mathbf{rec}}^{\mathbf{mean}}$	$\mathbf{t}_{\mathbf{rec}}^{\mathbf{max}}$
μ-POD-LSTM-ROM	0.0807 s	0.0892 s	0.3202 s	0.7609 s	0.8437 s	1.4095 s
POD-DL-ROM	0.1681 s	0.1857 s	0.4973 s	0.8727 s	0.9864 s	3.7111 s

5.4. Unsteady advection-diffusion-reaction time extrapolation

In this example, e tested the framework to assess its time extrapolation capabilities, both in the short term and in the long term. Such results in this context are remarkable, as traditional ROMs are not capable of time extrapolation for different parameter values than the ones used for training. Here we consider the same training parameters used before, while reducing the training time domain in order to consider only the first 60 time steps ($(0, T) = (0, 6\pi)$). A similar procedure is carried out for the test set, that includes snapshots for the same parametric instances $\boldsymbol{\mu} \in \mathcal{P}_{test}$ considered before, but a reduced time domain, chosen in order to contain just the last 60 time steps of the previous one, i.e., $(T_{in}, T_{fin}) = (4\pi, 10\pi)$. In this way, it is possible to test, on unseen parametric instances (in $\mathcal{P}_{test}$), a 40 time steps long time extrapolation.

The μ-POD-LSTM-ROM architecture used to build the μt-POD-LSTM-ROM framework is the same as the one considered before, while the t-POD-LSTM-ROM architecture considers $p = 10$ previous time steps in order to predict the following $k = 10$ time steps horizon. The number of trainable parameters for this latter network is $|\boldsymbol{\theta}_{ts}| = 151164$; its training took 1000 epochs (156 s).

The obtained results for an instance of the test set ($\boldsymbol{\mu} = (0.0035, 0.4750, 0.4750)$) are reported in Figure 8. The extrapolation precision is extremely high, even more considering that traditional projection-based methods do not allow for time extrapolation. Figure 9 reports the time evolution for a single DOF as well as the corresponding relative error evolution, while Table 6 reports some relevant quantities regarding the relative error for the time extrapolation task.

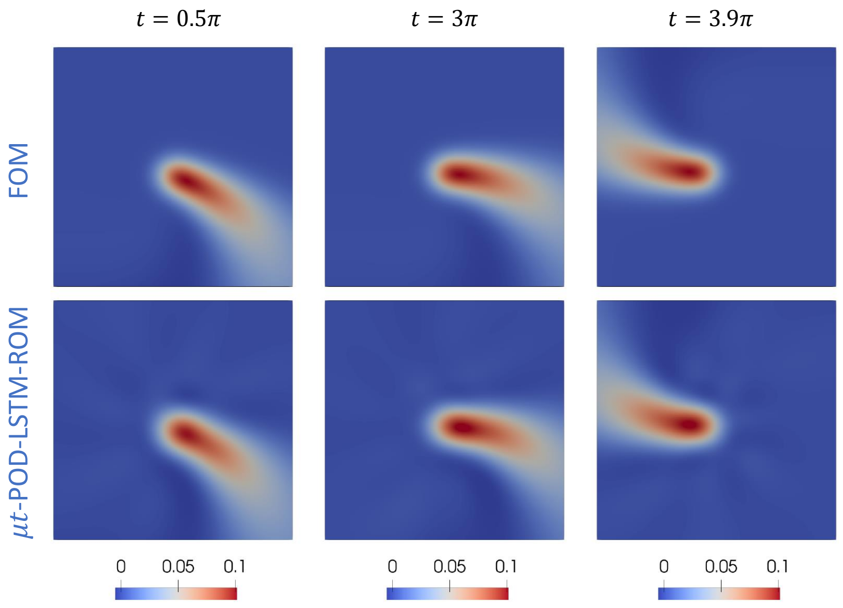

Figure 8. Test case 2–ADR equation. Simulation results for $\boldsymbol{\mu} = (0.0035, 0.4750, 0.4750)$. Top: FOM results, bottom: μt-POD-LSTM-ROM results.

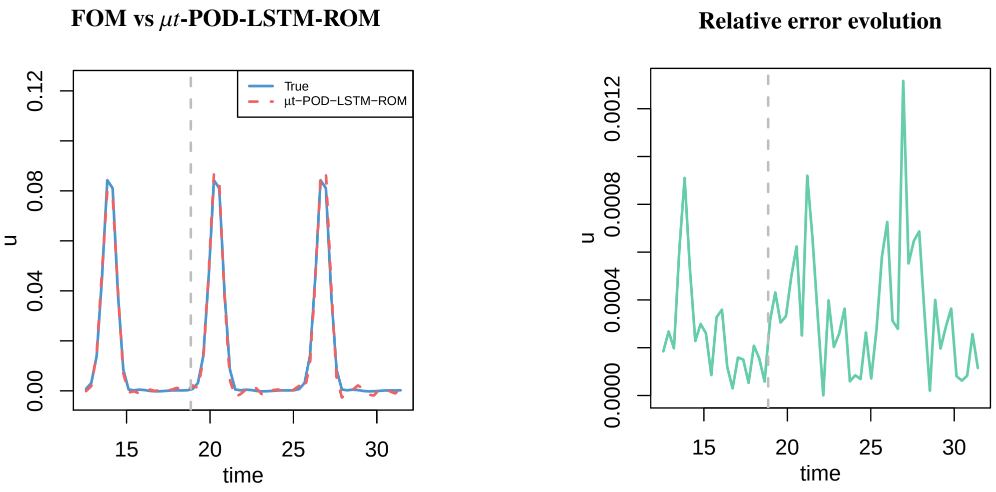

Figure 9. Test case 2–ADR equation. Solution (left) and relative error (right) time evolution obtained with the μt-POD-LSTM-ROM framework.

Table 6. Test case 2–ADR equation. Relative error indicators for the μt-POD-LSTM-ROM framework.

$\epsilon_{\mathbf{k}}^{\mathbf{mean}}$	$\epsilon_{\mathbf{k}}^{\mathbf{max}}$
$2.599 \cdot 10^{-4}$	$6.261 \cdot 10^{-3}$

The long-term extrapolation capabilities of the framework are extremely satisfying. In Figure 10 we report a long-term time extrapolation plot for the temporal evolution of a single DOF that considers a time window 16 times larger than the training domain. In general, over the entire test set performances are satisfying also in the long run, with some issues arising concerning scaling. A maximal systematic error of $\sim$ 20% arise when such time scales are considered, nonetheless the period is correctly reconstructed and there are no stability issues nor error explosion on the long term.

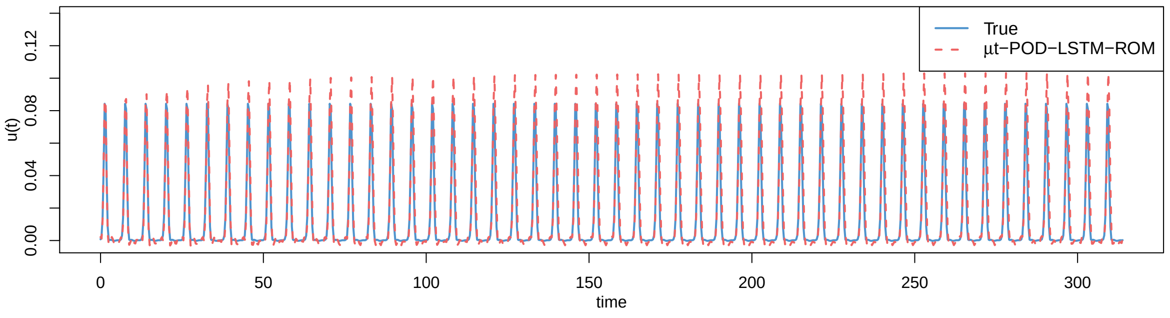

Figure 10. Test case 2–ADR equation. Long term (1000 time steps) time extrapolation for the solution at 8430th DOF using μt-POD-LSTM-ROM framework.

5.5. *Unsteady Navier-Stokes equations*

We finally focus on a fluid dynamics example–the well-known benchmark case of a two-dimensional unsteady flow past a cylinder–based on incompressible Navier-Stokes equations in the laminar regime. Our goal is to approximate the solution $\mathbf{u} = \mathbf{u}(\mathbf{x}, t; \mu)$ of the following problem:

$$\begin{cases} \rho \dfrac{\partial \mathbf{u}}{\partial t} + \rho \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot \boldsymbol{\sigma}(\mathbf{u}, p) = \mathbf{0} & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \nabla \cdot \mathbf{u} = 0 & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \mathbf{u} = \mathbf{0} & (\mathbf{x}, t) \in \Gamma_{D_1} \times (0, T), \\ \mathbf{u} = \mathbf{h} & (\mathbf{x}, t) \in \Gamma_{D_2} \times (0, T), \\ \boldsymbol{\sigma}(\mathbf{u}, p)\mathbf{n} = \mathbf{0} & (\mathbf{x}, t) \in \Gamma_N \times (0, T), \\ \mathbf{u}(0) = \mathbf{0} & \mathbf{x} \in \Omega, \ t = 0. \end{cases} \tag{5.5}$$

The reference domain represents a 2D pipe containing a circular obstacle with radius $r = 0.05$ centered in $\mathbf{x_{obs}} = (0.2, 0.2)$, i.e., $\Omega = (0, 2.2) \times (0, 0.41) \backslash \bar{B}_r(0.2, 0.2)$ (see Figure 11 for reference); this is a well-known benchmark test case already addressed in [33, 37]. The domain's boundary is $\partial\Omega = \Gamma_{D_1} \cup \Gamma_{D_2} \cup \Gamma_N \cup \partial B_{0.05}(0.2, 0.2)$, where $\Gamma_{D_1} = \{x_1 \in [0, 2.2], x_2 = 0\} \cup \{x_1 \in [0, 2.2], x_2 = 0.41\}$, $\Gamma_{D_2} = \{x_1 = 0, x_2 \in [0, 0.41]\}$, and $\Gamma_N = \{x_1 = 2.2, x_2 \in [0, 0.41]\}$; $\mathbf{n}$ denotes the outward directed versor, normal w.r.t. $\partial\Omega$. We consider $\rho = 1$ kg/m^3 to be the (constant) fluid density, and denote by

$$\boldsymbol{\sigma}(\mathbf{u}, p) = -p\mathbf{I} + 2\nu\boldsymbol{\epsilon}(\mathbf{u})$$

the stress tensor; here ν is the fluid's dynamic viscosity, while $\boldsymbol{\epsilon}(\mathbf{u})$ is the strain tensor,

$$\boldsymbol{\epsilon}(\mathbf{u}) = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T).$$

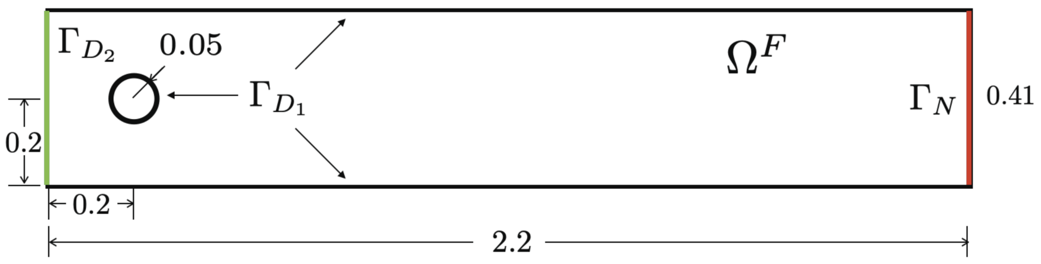

Figure 11. The 2D domain considered for the unsteady Navier-Stokes equations test case.

We assign no-slip boundary conditions on Γ_{D_1}, while a parabolic inflow profile

$$\mathbf{h}(\mathbf{x}, t; \mu) = \left(\frac{4U(t, \mu) x_2 (0.41 - x_2)}{0.41^2}, 0 \right), \qquad \text{with} \quad U(t; \mu) = \mu \sin(\pi t / 8), \tag{5.6}$$

is prescribed at the inlet Γ_{D_2}; zero-stress Neumann conditions are imposed at the outlet Γ_N. In this problem, we consider a single parameter ($n_\mu = 1$), $\mu \in \mathcal{P} = [1, 2]$, which is related with the magnitude of the inflow velocity and directly reflects on the Reynolds number, varying in the range Re $\in [66, 133]$.

Equation (5.5) has been discretized in space by means of linear-quadratic ($\mathbb{P}_2 - \mathbb{P}_1$), inf-sup stable, finite elements, and in time through a BDF of order 2 with semi-implicit treatment of the convective term over the time interval $(0, T)$ with $T = 6$, considering a time-step $\Delta t = 2 \times 10^{-3}$, leading to a computational time equal to 25000 s to solve the FOM.

For the sake of training speed we consider $N_t = 300$ uniformly distributed time instances and take $N_{train} = 21$ different parameter instances uniformly distributed over $\mathcal{P}$ and $N_{test} = 3$ parameters instances for testing, with $\mathcal{P}_{test} = \{1.025, 1.725, 1.975\}$, in order to perform testing considering both the center and the boundary of the parameters domain $\mathcal{P}$. In order to assess time extrapolation capabilities of μ-POD-LSTM-ROMs, we consider a testing time domain consisting in $N_t = 340$ time steps over the time interval $t \in (0, 6.8]$, resulting in an extrapolation time window of 13.3% w.r.t. the training interval. In this test case, scaling proved to deliver worse results in terms of accuracy and therefore no scaling is performed on POD-reduced data. We are interested in reconstructing the velocity field, for which the FOM dimension is equal to $N_h = 32446 \times 2 = 64892$, selecting $N = 256$ as dimension of the rPOD basis for each of the two velocity components.

We highlight the possibility, by using a μ-POD-LSTM-ROM, to reconstruct only the field of interest, i.e., the velocity $\mathbf{u}$, without the need of taking into account the approximation of the pressure p.

The μ-POD-LSTM-ROM architecture used for this benchmark case consists of a LSTM autoencoder built considering a single LSTM cell both for the encoder and the decoder, without further reducing the dimensionality of the POD-reduced vectors fed to the network. The hidden representation dimension was set to be $n = 200$, while the sequence length used for the training was fixed to $K = 20$. The number of trainable parameters for the network is $|\boldsymbol{\theta}| = 1365947$. The t-POD-LSTM-ROM architecture used for time extrapolation considers $p = 10$ time steps in the past in order to build the inference on $k = 10$ time steps in the future, exploiting a single LSTM cell autoencoder and a 3-layers feedforward regressor, for a total number of trainable weights $|\boldsymbol{\theta}_t| = 1616692$. The training of the μ-POD-LSTM-ROM network took 2415 epochs (total time: 5692 s), while the training of the t-POD-LSTM-ROM architecture took 325 epochs (total time: 694 s).

The obtained results for two instances of the test set ($\mu = 1.025$, near the border in $\mathcal{P}_{test}$ and no vortex shedding, and $\mu = 1.725$, central in $\mathcal{P}_{test}$ with vortex shedding) are reported in Figure 12 together with absolute and relative errors. A good accuracy is obtained considering time extrapolation, as the relative error reported in Table 7 shows. The mean relative error bootstrap 0.95 confidence interval is $C_{0.95}^{POD-LSTM-ROM} = [9.029 \cdot 10^{-5}, 1.053 \cdot 10^{-4}]$. Error indicator for this test case is $\epsilon_{rel} = 5.806 \cdot 10^{-2}$. The little magnitude of the error is especially remarkable considering *(a)* the failure of projection-based methods in time extrapolation, *(b)* the fact that architecture is just informed on the velocity field and thus cannot take advantage of data on pressure to increase the accuracy of the velocity field prediction and *(c)* the complexity of the problem at hand.

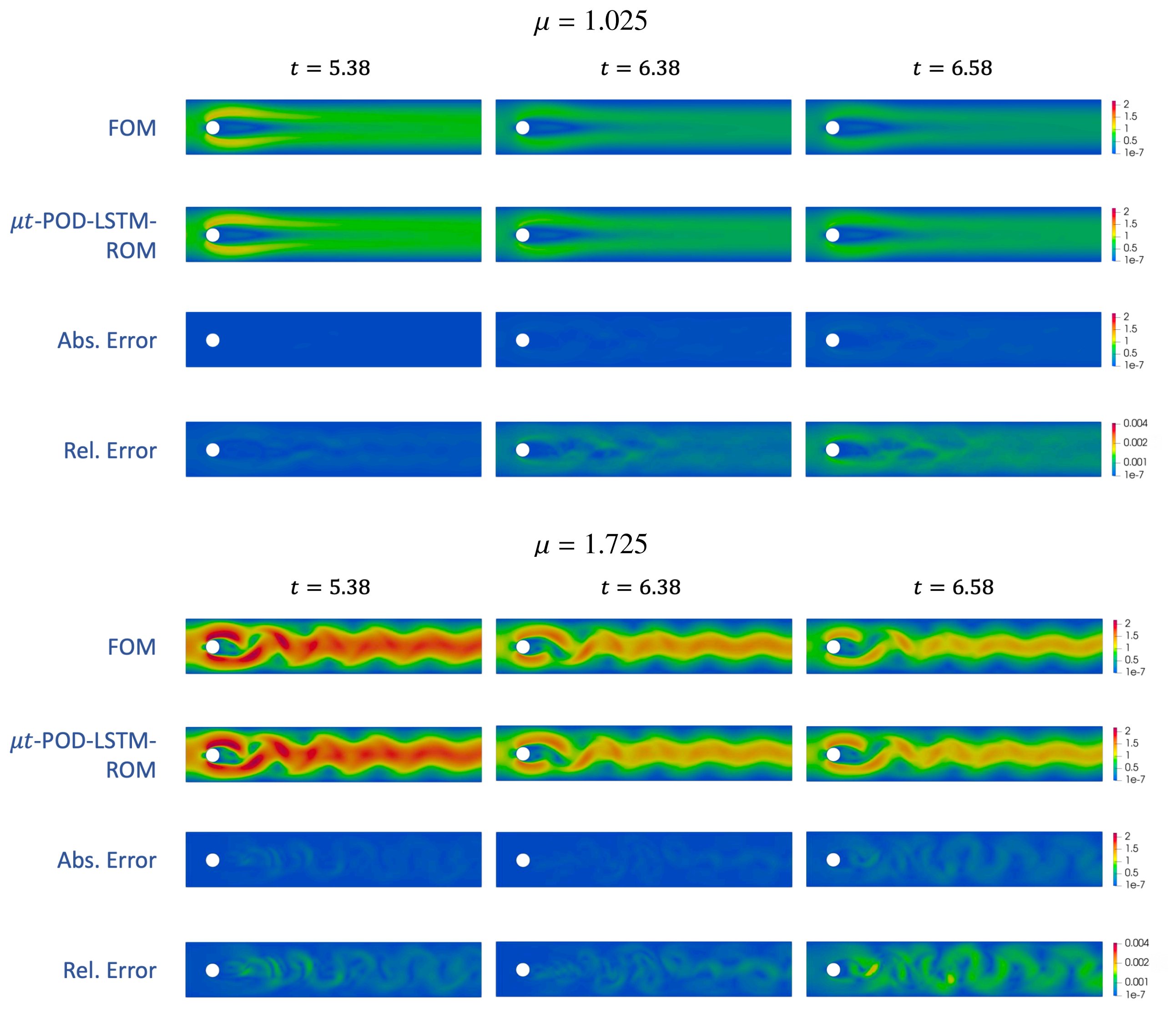

Figure 12. Test case 3–Navier-Stokes equations. μt-POD-LSTM-ROM velocity magnitude. Top: $Re = 68$–no vortex shedding; Bottom: $Re = 117$–vortex shedding. Time extrapolation starts at $t = 6$.

Table 7. Test case 3–Navier-Stokes equations. Relative error indicators for the μt-POD-LSTM-ROM framework applied to the unsteady Navier-Stokes problem.

$\boldsymbol{\epsilon}_{\mathbf{k}}^{\mathbf{mean}}$	$\boldsymbol{\epsilon}_{\mathbf{k}}^{\mathbf{max}}$
$9.808 \cdot 10^{-5}$	$5.460 \cdot 10^{-3}$

6. Conclusions

In this work we introduced μt-POD-LSTM-ROMs, a novel non-intrusive LSTM-based ROM framework that extends previous DL-based ROMs with time extrapolation capabilities. In addition, we improved the online performance of the already faster than real-time POD-DL-ROM framework. The strategy followed to pursue our goal splits the solution approximation problem into two parts: *(a)* the prediction of the solution for a new parameter instance and *(b)* the time extrapolation problem.

We therefore introduced two different LSTM-based architectures to address tasks *(a)* and *(b)* separately. In this way we have been able to replicate the extremely good approximation performances of POD-DL-ROMs on unseen parameters instances and–more importantly–we enriched the pre-existing DL-based ROMs with time extrapolation capabilities, otherwise hardly obtainable with POD-Galerkin ROMs.

We assessed the approximation accuracy, the computational performances and the time extrapolation capabilities on three different test cases: *(i)* a Lotka-Volterra 3 species prey-predator model, *(ii)* a linear unsteady advection-diffusion-reaction equation and *(iii)* the nonlinear unsteady Navier-Stokes equations with laminar flow. In particular, we observed extremely accurate time extrapolation capabilities, even on a longer term, for both periodic and simple aperiodic cases. Satisfactory time extrapolation capabilities ($\approx$ 15% of the training time domain) have also been obtained on complex test cases such as the considered benchmark test case in fluid dynamics. An immediate implication of this result is the possibility to produce the FOM snapshots on smaller time domains and therefore to observe a performance improvement in the offline phase. Furthermore, we obtained a 52.0% reduction of prediction times while maintaining the same order of magnitude on the relative error with respect of the (already extremely fast) POD-DL-ROM when applying the framework to advection-diffusion-reaction problems. This allowed us to obtain faster than real-time simulations of physical phenomena occurring in time scales of tenths of a second. Remarkably, the time performances of our novel framework allow for an "in local" training and testing, thus reducing deployment costs by avoiding the usage of cloud GPU clusters.

Ultimately, we provided a novel, fast, accurate and robust framework for the online approximation of parametric time-dependent PDEs, trainable also relying on black-box high-fidelity solvers and applicable to problems of interest in different realms.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgements

This work has been supported by Fondazione Cariplo, Italy, Grant n. 2019-4608. SF has also been supported by the PNRR-PE-AI FAIR project funded by the NextGeneration EU program. SF would like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the programme "The mathematical and statistical foundation of future data-driven engineering" where work on this paper was undertaken. This work was supported by EPSRC grant no EP/R014604. AM has also been supported by the National Group of Scientific Computing (GNCS) of INDAM - Istituto Nazionale di Alta Matematica.

Conflict of interest

The authors declare no conflicts of interest.

References

1. P. Benner, S. Gugercin, K. Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, *SIAM Rev.*, **57** (2015), 483–531. https://doi.org/10.1137/130932715
2. A. Quarteroni, A. Manzoni, F. Negri, *Reduced basis methods for partial differential equations: an introduction*, Springer, 2016. https://doi.org/10.1007/978-3-319-15431-2
3. P. Benner, A. Cohen, M. Ohlberger, K. Willcox, *Model reduction and approximation: theory and algorithms*, SIAM, 2017.
4. A. Quarteroni, A. Valli, *Numerical approximation of partial differential equations*, Springer, 1994. https://doi.org/10.1007/978-3-540-85268-1
5. A. Manzoni, An efficient computational framework for reduced basis approximation and a posteriori error estimation of parametrized Navier-Stokes flows, *ESAIM: Math. Modell. Numer. Anal.*, **48** (2014), 1199–1226. https://doi.org/10.1051/m2an/2014013
6. F. Ballarin, A. Manzoni, A. Quarteroni, G. Rozza, Supremizer stabilization of POD-Galerkin approximation of parametrized steady incompressible Navier-Stokes equations, *Int. J. Numer. Meth. Eng.*, **102** (2015), 1136–1161. https://doi.org/10.1002/nme.4772
7. N. Dal Santo, A. Manzoni, Hyper-reduced order models for parametrized unsteady Navier-Stokes equations on domains with variable shape, *Adv. Comput. Math.*, **45** (2019), 2463–2501. https://doi.org/10.1007/s10444-019-09722-9
8. C. Farhat, S. Grimberg, A. Manzoni, A. Quarteroni, Computational bottlenecks for PROMs: pre-computation and hyperreduction, In: P. Benner, S. Grivet-Talocia, A. Quarteroni, G. Rozza, W. Schilders, L. Silveira, *Model order reduction: volume 2: snapshot-based methods and algorithms*, Boston: De Gruyter, 2020, 181–244. https://doi.org/10.1515/9783110671490-005
9. G. Gobat, A. Opreni, S. Fresca, A. Manzoni, A. Frangi, Reduced order modeling of nonlinear microstructures through proper orthogonal decomposition, *Mech. Syst. Signal Process.*, **171** (2022), 108864. https://doi.org/10.1016/j.ymssp.2022.108864

10. I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Trans. Neur. Net.*, **9** (1998), 987–1000. https://doi.org/10.1109/72.712178
11. L. Aarts, P. van der Veer, Neural network method for solving partial differential equations, *Neural Process. Lett.*, **14** (2001), 261–271. https://doi.org/10.1023/A:1012784129883
12. K. Hornik, Approximation capabilities of multilayer feedforward networks, *Neural Networks*, **4** (1991), 251–257. https://doi.org/10.1016/0893-6080(91)90009-T
13. Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks, *Eur. J. Appl. Math.*, **32** (2021), 421–435. https://doi.org/10.1017/S0956792520000182
14. C. Michoski, M. Milosavljević, T. Oliver, D. Hatch, Solving differential equations using deep neural networks, *Neurocomputing*, **399** (2020), 193–212. https://doi.org/10.1016/j.neucom.2020.02.015
15. J. Berg, K. Nyström, Data-driven discovery of PDEs in complex datasets, *J. Comput. Phys.*, **384** (2019), 239–252. https://doi.org/10.1016/j.jcp.2019.01.036
16. M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.*, **378** (2019), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
17. M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, *J. Mach. Learn. Res.*, **19** (2018), 932–955. https://doi.org/10.5555/3291125.3291150
18. J. A. A. Opschoor, P. C. Petersen, C. Schwab, Deep ReLU networks and high-order finite element methods, *Anal. Appl.*, **18** (2020), 715–770. https://doi.org/10.1142/S0219530519410136
19. G. Kutyniok, P. Petersen, M. Raslan, R. Schneider, A theoretical analysis of deep neural networks and parametric PDEs, *Constr. Approx.*, **55** (2021), 73–125. https://doi.org/10.1007/s00365-021-09551-4
20. D. Yarotsky, Error bounds for approximations with deep relu networks, *Neural Networks*, **94** (2017), 103–114. https://doi.org/10.1016/j.neunet.2017.07.002
21. N. Franco, A. Manzoni, P. Zunino, A deep learning approach to reduced order modelling of parameter dependent partial differential equations, *Math. Comp.*, **92** (2023), 483–524. https://doi.org/10.1090/mcom/3781
22. T. De Ryck, S. Mishra, Generic bounds on the approximation error for physics-informed (and) operator learning, *arXiv*, 2022. https://doi.org/10.48550/arXiv.2205.11393
23. N. Kovachki, S. Lanthaler, S. Mishra, On universal approximation and error bounds for Fourier neural operators, *J. Mach. Learn. Res.*, **22** (2021), 13237–13312. https://doi.org/10.5555/3546258.3546548
24. S. Lanthaler, S. Mishra, G. E. Karniadakis, Error estimates for DeepONets: a deep learning framework in infinite dimensions, *Trans. Math. Appl.*, **6** (2022), tnac001. https://doi.org/10.1093/imatrm/tnac001
25. M. Guo, J. S. Hesthaven, Reduced order modeling for nonlinear structural analysis using gaussian process regression, *Comput. Methods Appl. Mech. Eng.*, **341** (2018), 807–826. https://doi.org/10.1016/j.cma.2018.07.017

26. M. Guo, J. S. Hesthaven, Data-driven reduced order modeling for time-dependent problems, *Comput. Methods Appl. Mech. Eng.*, **345** (2019), 75–99. https://doi.org/10.1016/j.cma.2018.10.029

27. J. S. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, *J. Comput. Phys.*, **363** (2018), 55–78. https://doi.org/10.1016/j.jcp.2018.02.037

28. S. Pawar, S. E. Ahmed, O. San, A. Rasheed, Data-driven recovery of hidden physics in reduced order modeling of fluid flows, *Phys. Fluids*, **32** (2020), 036602. https://doi.org/10.1063/5.0002051

29. B. A. Freno, K. T. Carlberg, Machine-learning error models for approximate solutions to parameterized systems of nonlinear equations, *Comput. Methods Appl. Mech. Eng.*, **348** (2019) 250–296. https://doi.org/10.1016/j.cma.2019.01.024

30. E. J. Parish, K. T. Carlberg, Time-series machine learning error models for appproximate solutions to dynamical systems, 15th *National Congress of Computational Mechanics*, 2019.

31. K. Lee, K. T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, *J. Comput. Phys.*, **404** (2020), 108973. https://doi.org/10.1016/j.jcp.2019.108973

32. S. Fresca, L. Dedè, A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs, *J. Sci. Comput.*, **87** (2021), 61. https://doi.org/10.1007/s10915-021-01462-7

33. S. Fresca, A. Manzoni, POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition, *Comput. Methods Appl. Mech. Eng.*, **388** (2021), 114181. https://doi.org/10.1016/j.cma.2021.114181

34. N. Halko, P. G. Martinsson, J. A. Tropp, Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, *SIAM Rev.*, **53** (2011), 217–288. https://doi.org/10.1137/090771806

35. S. Fresca, A. Manzoni, L. Dedè, A. Quarteroni, Deep learning-based reduced order models in cardiac electrophysiology, *PLoS One*, **15** (2020), e0239416. https://doi.org/10.1371/journal.pone.0239416

36. S. Fresca, A. Manzoni, L. Dedè, A. Quarteroni, POD-enhanced deep learning-based reduced order models for the real-time simulation of cardiac electrophysiology in the left atrium, *Front. Physiol.*, **12** (2021), 679076. https://doi.org/10.3389/fphys.2021.679076

37. S. Fresca, A. Manzoni, Real-time simulation of parameter-dependent fluid flows through deep learning-based reduced order models, *Fluids*, **6** (2021), 259. https://doi.org/10.3390/fluids6070259

38. S. Fresca, G. Gobat, P. Fedeli, A. Frangi, A. Manzoni, Deep learning-based reduced order models for the real-time simulation of the nonlinear dynamics of microstructures, *Int. J. Numer. Methods Eng.*, **123** (2022), 4749–4777. https://doi.org/10.1002/nme.7054

39. S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural Comput.*, **9** (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

40. F. Gers, J. Schmidhuber, F. Cummins, Learning to forget: continual prediction with LSTM, *Neural Comput.*, **12** (2000), 2451–71. https://doi.org/10.1162/089976600300015015

41. P. Sentz, K. Beckwith, E. C. Cyr, L. N. Olson, R. Patel, Reduced basis approximations of parameterized dynamical partial differential equations via neural networks, *arXiv*, 2021. https://doi.org/10.48550/arXiv.2110.10775

42. R. Maulik, B. Lusch, P. Balaprakash, Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders, *Phys. Fluids*, **33** (2021), 037106. https://doi.org/10.1063/5.0039986

43. J. Xu, K. Duraisamy, Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics, *Comput. Methods Appl. Mech. Eng.*, **372** (2020), 113379. https://doi.org/10.1016/j.cma.2020.113379

44. N. T. Mücke, S. M. Bohté, C. W. Oosterlee, Reduced order modeling for parameterized time-dependent pdes using spatially and memory aware deep learning, *J. Comput. Sci.*, **53** (2021), 101408. https://doi.org/10.1016/j.jocs.2021.101408

45. Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, H. Zhang, Deep learning with long short-term memory for time series prediction, *IEEE Commun. Mag.*, **57** (2019), 114–119. https://doi.org/10.1109/MCOM.2019.1800155

46. R. Maulik, B. Lusch, P. Balaprakash, Non-autoregressive time-series methods for stable parametric reduced-order models, *Phys. Fluids*, **32** (2020), 087115. https://doi.org/10.1063/5.0019884

47. N. Srivastava, E. Mansimov, R. Salakhutdinov, Unsupervised learning of video representations using LSTMs, *ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning*, **37** (2015), 843–852.

48. P. Drineas, R. Kannan, M. W. Mahoney, Fast Monte Carlo algorithms for matrices II: computing a low-rank approximation to a matrix, *SIAM J. Comput.*, **36** (2006), 158–183. https://doi.org/10.1137/S0097539704442696

49. M. Sangiorgio, F. Dercole, Robustness of LSTM neural networks for multi-step forecasting of chaotic time series, *Chaos Soliton. Fract.*, **39** (2020), 110045. https://doi.org/10.1016/j.chaos.2020.110045

50. S. Du, T. Li, S. Horng, Time series forecasting using sequence-to-sequence deep learning framework, *2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, 2018, 171–176. https://doi.org/10.1109/PAAP.2018.00037

51. W. Zucchini, I. Macdonald, Hidden Markov models for time series: an introduction using R, 1 Ed., New York: Chapman and Hall/CRC, 2009. https://doi.org/10.1201/9781420010893

52. P. Dostál, Forecasting of time series with fuzzy logic, In: I. Zelinka, G. Chen, O. E. Rössler, V. Snasel, A. Abraham, *Nostradamus 2013: prediction, modeling and analysis of complex systems*, Heidelberg: Springer, **210** (2013), 155–161. https://doi.org/10.1007/978-3-319-00542-3_16

53. F. A. Gers, D. Eck, J. Schmidhuber, Applying LSTM to time series predictable through time-window approaches, In: G. Dorffner, H. Bischof, K. Hornik, *Artificial neural networks — ICANN 2001*, Lecture Notes in Computer Science, Springer, **2130** (2001), 669–676. https://doi.org/10.1007/3-540-44668-0_93

54. S. Siami-Namini, N. Tavakoli, A. Siami Namin, A comparison of ARIMA and LSTM in forecasting time series, *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, 1394–1401. https://doi.org/10.1109/ICMLA.2018.00227

55. R. T. Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, *NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, 6572–6583. https://doi.org/10.5555/3327757.3327764

56. S. Massaroli, M. Poli, J. Park, A. Yamashita, H. Asama, Dissecting neural ODEs, *arXiv*, 2021. https://doi.org/10.48550/arXiv.2002.08071

57. P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, *Proc. R. Soc. A: Math. Phys. Eng. Sci.*, **474** (2018), 1–20. https://doi.org/10.1098/rspa.2017.0844

58. D. P. Kingma, J. Ba, ADAM: a method for stochastic optimization, *3rd International Conference for Learning Representations*, San Diego, 2015.

59. I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT Press, 2016. Available from: `http://www.deeplearningbook.org`.

60. D. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), *arXiv*, 2015. https://doi.org/10.48550/arXiv.1511.07289

61. K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, 1026–1034.

62. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et al., TensorFlow: large-scale machine learning on heterogeneous systems, 2015. Available from: `https://www.tensorflow.org`.

63. F. Negri, redbkit v2.2, 2017. Available from: `https://github.com/redbKIT/redbKIT`.

64. J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, *J. Mach. Learn. Res.*, **13** (2012), 281–305. https://doi.org/10.5555/2188385.2188395

65. S. Chaturantabut, D. C. Sorensen, Discrete empirical interpolation for nonlinear model reduction, *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 2009, 4316–4321. https://doi.org/10.1109/CDC.2009.5400045

66. C. M. Bishop, *Neural networks for pattern recognition*, Oxford University Press, Inc., 1995.

Appendix

In this appendix, we report in extensive form the algorithms used to train and test the μt-POD-LSTM-ROM framework. In particular, Algorithm 1 and Algorithm 2 describe the procedures used for training the μ-POD-LSTM-ROM and t-POD-LSTM-ROM architectures respectively, while Algorithm 3 and Algorithm 4 outline the testing stage of the two architectures. Finally, Algorithm 5 specifies the workflow required by the μt-POD-LSTM-ROM framework in order to obtain the fast approximation of a parameterized PDE solution with time extrapolation capabilities.

Algorithm 1 μ-POD-LSTM-ROM training algorithm.

```
Input: Parameter matrix $\mathbf{M} \in \mathbb{R}^{(n_\mu+1)\times N_{train}N_t}$, snapshot matrix $\mathbf{S} \in \mathbb{R}^{N_h \times N_{train}N_t}$, sequence length $K > 1$, validation split $\alpha > 0$, starting learning rate $\eta > 0$, batch size $\dim_{batch}$, maximum number of epochs $N_{ep}$, loss parameter $\omega_h$.
Output: Optimal model parameters
    $\boldsymbol{\theta}^{*,\mu} = (\boldsymbol{\theta}^{*,\mu}_{FFNN}, \boldsymbol{\theta}^{*,\mu}_{enc}, \boldsymbol{\theta}^{*,\mu}_{dec})$.
1: Compute rPOD basis matrix $\mathbf{V}_N$
2: Compute the POD reduced snapshot matrix
    $\mathbf{U}_N = [\mathbf{u}_1|\dots|\mathbf{u}_{N_{train}N_t}]^T = [\mathbf{V}_N^T\mathbf{u}_{h,1}|\dots|\mathbf{V}_N^T\mathbf{u}_{h,N_{train}N_t}]^T$
3: Assemble the base tensor $\mathbf{T} \in \mathbb{R}^{N_{train}(N_t-K)\times N\times K}$ with $\mathbf{T}(i,j,k) = (\mathbf{u}_N(t_{\alpha_i} + k\Delta t, \boldsymbol{\mu}_{\beta_i}))_j$ with $\alpha_i = i\mathrm{mod}(N_t - K), \beta_i = \frac{i-\alpha_i}{N_t-K}$ and $(\cdot)_j$ denoting the extraction of the $j^{th}$ component from a vector.
4: Assemble the base parameters tensor $\mathbf{L} \in \mathbb{R}^{N_{train}(N_t-K)\times(n_\mu+1)\times K}$ with $\mathbf{L}(i,j,k) = (\mathbf{M}[\alpha_i + k, :])_j$ with $\alpha_i = i\mathrm{mod}(N_t - K), \beta_i = \frac{i-\alpha_i}{N_t-K}$ and $(\cdot)_j$ denoting the extraction of the $j^{th}$ component from a vector.
5: Randomly shuffle $\mathbf{T}$ and $\mathbf{L}$
6: Randomly sample $\alpha N_{train}(N_t - K)$ indices from $\mathbf{I} = \{0,\dots,N_{train}(N_t - K) - 1\}$ and collect them in the vector $val\_idxs$. Build $train\_idxs = I \setminus val\_idxs$
7: Split data in $\mathbf{T} = [\mathbf{T}^{train}, \mathbf{T}^{val}]$ and $\mathbf{L} = [\mathbf{L}^{train}, \mathbf{L}^{val}]$ (with $\mathbf{T}^{train/val} = \mathbf{T}[train/val\_idxs, :, :], \mathbf{L}^{train/val} = \mathbf{L}[train/val\_idxs, :, :]$)
8: Optionally normalize data in $\mathbf{T}$
9: Randomly initialize $\boldsymbol{\theta}^0 = (\boldsymbol{\theta}^0_{FFNN}, \boldsymbol{\theta}^0_{enc}, \boldsymbol{\theta}^0_{dec})$
10: $n_e = 0$
11: while ($\neg$early-stopping and $n_e \leq N_{ep}$) do
12:     for $b = 1 : N_{mb}$ do
13:         Sample a minibatch $(\mathbf{T}^{batch}_{N,K}, \mathbf{L}^{batch}) \subseteq (\mathbf{T}^{train}_{N,K}, \mathbf{L}^{train})$
14:         $\widetilde{\mathbf{T}}^{batch}_{n,K}(\boldsymbol{\theta}^{N_{mb}n_e+b}_{enc}) = \boldsymbol{\lambda}^{enc}_n(\mathbf{T}^{batch}_{N,K}; \boldsymbol{\theta}^{N_{mb}n_e+b}_{enc})$
15:         $\mathbf{T}^{batch}_{n,K}(\boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN}) = \boldsymbol{\phi}^{FFNN}_n(\mathbf{L}^{batch}; \boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN})$
16:         $\widetilde{\mathbf{T}}^{batch}_{N,K}(\boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN}, \boldsymbol{\theta}^{N_{mb}n_e+b}_{dec}) = \boldsymbol{\lambda}^{dec}_N(\mathbf{T}^{batch}_{n,K}(\boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN}); \boldsymbol{\theta}^{N_{mb}n_e+b}_{dec})$
17:         Accumulate loss (3.9) on $(\mathbf{T}^{batch}_{N,K}, \mathbf{L}^{batch})$ and compute $\widehat{\nabla}_\theta\mathcal{J}$
18:         $\boldsymbol{\theta}^{N_{mb}n_e+b+1} = \mathrm{ADAM}(\eta, \widehat{\nabla}_\theta\mathcal{J}, \boldsymbol{\theta}^{N_{mb}n_e+b})$
19:     end for
20:     Repeat instructions 13–18 on $(\mathbf{T}^{val}_{N,K}, \mathbf{L}^{val})$ with the updated weights $\boldsymbol{\theta}^{N_{mb}n_e+b+1}$
21:     Accumulate loss (3.9) on $(\mathbf{T}^{val}_{N,K}, \mathbf{L}^{val})$ to evaluate early-stopping criterion
22:     $n_e = n_e + 1$
23: end while
```

Algorithm 2 t-POD-DL-ROM training algorithm.

Input: Parameter matrix $\mathbf{M} \in \mathbb{R}^{(n_\mu)\times N_{train}N_t}$, POD reduced snapshot matrix $\mathbf{U}_N \in \mathbb{R}^{N\times N_{train}N_t}$, sequence length $K > 2$, prediction horizon $1 \leq k < K$, validation split $\alpha > 0$, starting learning rate $\eta > 0$, batch size $\dim_{batch}$, maximum number of epochs N_{ep}, loss parameter ω_h.

Output: Optimal model parameters

$$\boldsymbol{\theta}^{*,t} = (\boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{enc}, \boldsymbol{\theta}^{*,t}_{dec}).$$

```
1: Assemble the base tensor
   $$\mathbf{T} \in \mathbb{R}^{N_{train}(N_t-K)\times N\times K} \text{ with } \mathbf{T}(i,j,k) = (\mathbf{u}_N(t_{\alpha_i} + k\Delta t, \boldsymbol{\mu}_{\beta_i}))_j$$
   with $\alpha_i = i \bmod (N_t - K)$, $\beta_i = \frac{i-\alpha_i}{N_t-K}$ and $(\cdot)_j$ denoting the extraction of the $j^{th}$ component from a vector.
2: Assemble the base parameters tensor
   $$\mathbf{L} \in \mathbb{R}^{N_{train}(N_t-K)\times n_\mu\times K} \text{ with } \mathbf{L}(i,j,k) = (\mathbf{M}[\alpha_i + k, : (n_\mu)])_j$$
   with $\alpha_i = i \bmod (N_t - K)$, $\beta_i = \frac{i-\alpha_i}{N_t-K}$ and $(\cdot)_j$ denoting the extraction of the $j^{th}$ component from a vector.
3: Randomly shuffle by the first dimension **T** and **L**
4: Randomly sample $\alpha N_{train}(N_t - K)$ indices from $\mathbf{I} = \{0, \dots, N_{train}(N_t - K) - 1\}$ and collect them in the vector *val_idxs*; build *train_idxs = I \ val_idxs*
5: Split data in $\mathbf{T} = [\mathbf{T}^{train}, \mathbf{T}^{val}]$, $\mathbf{L} = [\mathbf{L}^{train}, \mathbf{L}^{val}]$ (with $\mathbf{T}^{train/val} = \mathbf{T}[train/val\_idxs, :, :], \mathbf{L}^{train/val} = \mathbf{L}[train/val\_idxs, :, :]$)
6: Optionally normalize data in **T**
7: Randomly initialize $\boldsymbol{\theta}^0 = (\boldsymbol{\theta}^0_{FFNN}, \boldsymbol{\theta}^0_{enc}, \boldsymbol{\theta}^0_{dec})$
8: $n_e = 0$
9: **while** ($\neg$early-stopping **and** $n_e \leq N_{ep}$) **do**
10:     **for** $b = 1 : N_{mb}$ **do**
11:         Sample a minibatch $(\mathbf{T}^{batch}_{N,K}, \mathbf{L}^{batch}) \subseteq (\mathbf{T}^{train}_{N,K}, \mathbf{L}^{train})$
12:         Consider $\mathbf{T}^{batch}_0 = \mathbf{T}^{train}_{N,K}[:, :, : (K-k)]$ and $\mathbf{T}^{batch}_1 = \mathbf{T}^{train}_{N,K}[:, :, (K-k) : K)]$
13:         $\mathbf{R}^{batch}_0(\boldsymbol{\theta}^{N_{mb}n_e+b}_{enc}) = \boldsymbol{\lambda}^{enc}_n(\mathbf{T}^{batch}_0; \boldsymbol{\theta}^{N_{mb}n_e+b}_{enc})$
14:         $\mathbf{R}^{batch}_1(\boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN1}) = \boldsymbol{\phi}(\mathbf{L}^{batch}; \boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN1})$
15:         $\mathbf{H}^{batch}_n(\boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN}, \boldsymbol{\theta}^{N_{mb}n_e+b}_{enc}) = \boldsymbol{\phi}'([\mathbf{R}^{batch}_1, \mathbf{R}^{batch}_0]; \boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN2})$
16:         $\widetilde{\mathbf{T}}^{batch}_{N,k}(\boldsymbol{\theta}^{N_{mb}n_e+b}_{enc}, \boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN}, \boldsymbol{\theta}^{N_{mb}n_e+b}_{dec}) = \boldsymbol{\lambda}^{dec}_N(\mathbf{H}^{batch}_n(\boldsymbol{\theta}^{N_{mb}n_e+b}_{FFNN}, \boldsymbol{\theta}^{N_{mb}n_e+b}_{enc}); \boldsymbol{\theta}^{N_{mb}n_e+b}_{dec})$
17:         Accumulate loss (4.7) on $(\mathbf{T}^{batch}_1, \mathbf{L}^{batch})$ and compute $\widehat{\nabla}_\theta \mathcal{J}$
18:         $\boldsymbol{\theta}^{N_{mb}n_e+b+1} = \text{ADAM}(\eta, \widehat{\nabla}_\theta \mathcal{J}, \boldsymbol{\theta}^{N_{mb}n_e+b})$
19:     **end for**
20:     Repeat instructions 11–18 on $(\mathbf{T}^{val}_{N,K}, \mathbf{L}^{val})$ with the updated weights $\boldsymbol{\theta}^{N_{mb}n_e+b+1}$
21:     Accumulate loss (4.7) on $(\mathbf{T}^{val}_{N,K}, \mathbf{L}^{val})$ to evaluate early-stopping criterion
22:     $n_e = n_e + 1$
23: **end while**
```

Algorithm 3 μ-POD-LSTM-ROM testing algorithm.

Input: Testing parameter matrix $\mathbf{M}^{test} \in \mathbb{R}^{(n_\mu+1)\times N_{test}N_t}$, rPOD basis matrix $\mathbf{V}_N$, optimal model parameters $(\boldsymbol{\theta}^{*,\mu}_{FFNN}, \boldsymbol{\theta}^{*,\mu}_{dec})$.

Output: ROM approximation matrix

$$\widetilde{\mathbf{S}}_h \in \mathbb{R}^{N_h\times(N_{test}N_t)}.$$

```
1:  Build the reduced testing parameter matrix $\mathbf{M}^{test}_{red} \in \mathbb{R}^{(n_\mu+1)\times N_{test}\cdot N_t/K}$ s.t. $\mathbf{M}^{test}_{red}[:,i] = M^{test}[:,K\cdot i]$ $\forall i \in \{0,\dots,N_{test}\cdot N_t/K-1\}$
2:  Load $\boldsymbol{\theta}^{*,\mu}_{FFNN}$ and $\boldsymbol{\theta}^{*,\mu}_{dec}$
3:  $\mathbf{T}_{n,K}(\boldsymbol{\theta}^{*,\mu}_{FFNN}) = \boldsymbol{\phi}^{FFNN}_n(\mathbf{M}^{test}_{red}; \boldsymbol{\theta}^{*,\mu}_{FFNN})$
4:  $\widetilde{\mathbf{T}}_{N,K}(\boldsymbol{\theta}^{*,\mu}_{FFNN}, \boldsymbol{\theta}^{*,\mu}_{dec}) = \boldsymbol{\lambda}^{dec}_N(\mathbf{T}_n(\boldsymbol{\theta}^{*,\mu}_{FFNN}); \boldsymbol{\theta}^{*,\mu}_{dec})$
5:  Reshape $\widetilde{\mathbf{T}}_{N,K}(\boldsymbol{\theta}^{*,\mu}_{FFNN}, \boldsymbol{\theta}^{*,\mu}_{dec}) \in \mathbb{R}^{N_h\times N_{test}\cdot N_t/K\times K}$ in $\widetilde{\mathbf{S}}_N(\boldsymbol{\theta}^{*,\mu}_{FFNN}, \boldsymbol{\theta}^{*,\mu}_{dec}) \in \mathbb{R}^{N_h\times N_{test}N_t}$
6:  $\widetilde{\mathbf{S}}_h = \mathbf{V}_N\widetilde{\mathbf{S}}_N$
```

Algorithm 4 t-POD-LSTM-ROM testing algorithm.

Input: Testing parameter matrix (without times) $\mathbf{M}^{test}_{-} \in \mathbb{R}^{n_\mu\times N_{test}}$, rPOD basis matrix $\mathbf{V}_N$, μ-POD-LSTM-ROM reduced approximation matrix optimal model parameters $\widetilde{\mathbf{S}}_N = \mathbf{V}^T_N\widetilde{\mathbf{S}}_h \in \mathbb{R}^{N\times(N_{test}N_t)}$, extrapolation starting point $1 \le t_{ext} \le N_t$, extrapolation length $N_{ext} \ge 1$, prediction horizon $1 \le k < K$ (with K being the sequence length used for training), optimal training parameters $(\boldsymbol{\theta}^{*,t}_{enc}, \boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{dec})$.

Output: ROM extrapolation matrix

$$\widetilde{\mathbf{E}}_h \in \mathbb{R}^{N_h\times(N_{test}\cdot N_{ext}k)}.$$

```
1:  Allocate memory for $\widetilde{\mathbf{E}}_{N,K} \in \mathbb{R}^{N_{test}\times N\times((N_{ext}-1)k+K)}$
2:  Initialize $\widetilde{\mathbf{E}}_{N,K}$ by setting $\widetilde{\mathbf{E}}_{N,K}[:,i,:(K-k)] = \widetilde{\mathbf{S}}_N[:,(iN_t+t_{ext}-K+k):(iN_t+t_{ext})]$ $\forall i \in \{0,\dots,N_{test}-1\}$
3:  Load $\boldsymbol{\theta}^{*,t}_{enc}$, $\boldsymbol{\theta}^{*,t}_{FFNN}$ and $\boldsymbol{\theta}^{*,t}_{dec}$
4:  $c = 1$
5:  for $j = 1 : N_{ext}$ do
6:      $\mathbf{R}^{test,0}(\boldsymbol{\theta}^{*,t}_{enc}) = \boldsymbol{\lambda}^{enc}_n(\widetilde{\mathbf{E}}_{N,K}[:,:,c:(c+K-k)]; \boldsymbol{\theta}^{*,t}_{enc})$
7:      $\mathbf{R}^{test,1}(\boldsymbol{\theta}^{*,t}_{FFNN1}) = \boldsymbol{\phi}(\mathbf{M}^{test}_{-}; \boldsymbol{\theta}^{*,t}_{FFNN1})$
8:      $\mathbf{H}^{test}_n(\boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{enc}) = \boldsymbol{\phi}'([\mathbf{R}^{test,1}, \mathbf{R}^{test,0}]; \boldsymbol{\theta}^{*,t}_{FFNN2})$
9:      $\widetilde{\mathbf{E}}_{N,K}(\boldsymbol{\theta}^{*,t}_{enc}, \boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{dec})[:,:,(c+K-k):(c+K)] = \boldsymbol{\lambda}^{dec}_N(\mathbf{H}^{test}_n(\boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{enc}); \boldsymbol{\theta}^{*,t}_{dec})$
10:     $c = c + k$
11: end for
12: Consider the matrix containing extrapolation results only, i.e., $\widetilde{\mathbf{E}}^{ext}_{N,K}(\boldsymbol{\theta}^{*,t}_{enc}, \boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{dec}) = \widetilde{\mathbf{E}}_{N,K}(\boldsymbol{\theta}^{*,t}_{enc}, \boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{dec})[:,:,(K-k):]$
13: Reshape $\widetilde{\mathbf{E}}^{ext}_{N,K}(\boldsymbol{\theta}^{*,t}_{enc}, \boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{dec}) \in \mathbb{R}^{N_{test}\times N\times N_{ext}k}$ in $\widetilde{\mathbf{E}}_N(\boldsymbol{\theta}^{*,t}_{FFNN}, \boldsymbol{\theta}^{*,t}_{dec}) \in \mathbb{R}^{N\times(N_{test}\cdot N_{ext}k)}$
14: $\widetilde{\mathbf{E}}_h = \mathbf{V}_N\widetilde{\mathbf{E}}_N$
```

Algorithm 5 μt-POD-LSTM-ROM training-testing algorithm.

Input: The same inputs as Algorithms 1–4.
Output: Time extended ROM approximation matrix

$$\widetilde{\mathbf{S}}_h^{ext} \in \mathbb{R}^{N_h \times N_{test}(t_{ext}+N_{ext}k)}.$$

```
1: Train $\mu$-POD-LSTM-ROM architecture according to Algorithm 1
2: Train $t$-POD-LSTM-ROM architecture according to Algorithm 2
3: Load $(\boldsymbol{\theta}_{enc}^{*,\mu}, \boldsymbol{\theta}_{FFNN}^{*,\mu}, \boldsymbol{\theta}_{dec}^{*,\mu})$ and $(\boldsymbol{\theta}_{enc}^{*,t}, \boldsymbol{\theta}_{FFNN}^{*,t}, \boldsymbol{\theta}_{dec}^{*,t})$
4: Obtain $\widetilde{\mathbf{S}}_h \in \mathbb{R}^{N_h \times N_{test}N_t}$ by the procedure described in Algorithm 3
5: Obtain $\widetilde{\mathbf{E}}_h \in \mathbb{R}^{N_h \times (N_{test} \cdot N_{ext}k)}$ by the procedure described in Algorithm 4
6: Assemble the time extended ROM approximation matrix $\widetilde{\mathbf{S}}_h^{ext}$ by concatenating by column part of
   the previous results as in $\widetilde{\mathbf{S}}_h^{ext} = \widetilde{\mathbf{S}}_h[:, : t_{ext}] \oplus \widetilde{\mathbf{E}}_h$
```