Data-Centric Engineering (2021), 2: e13
doi:10.1017/dce.2021.15

RESEARCH ARTICLE

Model order reduction based on Runge–Kutta neural networks

Qinyu Zhuang[1,*], Juan Manuel Lorenzi[1], Hans-Joachim Bungartz[2] and Dirk Hartmann[1]

[1]Technology, Siemens AG, Bayern, Germany
[2]Chair of Scientific Computing, Technical University of Munich, Bayern, Germany
*Corresponding author. E-mail: qinyu.zhuang@siemens.com

Received: 06 February 2021; **Revised:** 10 June 2021; **Accepted:** 11 July 2021

Keywords: Dynamic parameter sampling; explicit Euler neural network; model order reduction; multilayer perceptron; Runge–Kutta neural network

Abstract

Model order reduction (MOR) methods enable the generation of real-time-capable digital twins, with the potential to unlock various novel value streams in industry. While traditional projection-based methods are robust and accurate for linear problems, incorporating machine learning to deal with nonlinearity becomes a new choice for reducing complex problems. These kinds of methods are independent to the numerical solver for the full order model and keep the nonintrusiveness of the whole workflow. Such methods usually consist of two steps. The first step is the dimension reduction by a projection-based method, and the second is the model reconstruction by a neural network (NN). In this work, we apply some modifications for both steps respectively and investigate how they are impacted by testing with three different simulation models. In all cases Proper orthogonal decomposition is used for dimension reduction. For this step, the effects of generating the snapshot database with constant input parameters is compared with time-dependent input parameters. For the model reconstruction step, three types of NN architectures are compared: multilayer perceptron (MLP), explicit Euler NN (EENN), and Runge–Kutta NN (RKNN). The MLPs learn the system state directly, whereas EENNs and RKNNs learn the derivative of system state and predict the new state as a numerical integrator. In the tests, RKNNs show their advantage as the network architecture informed by higher-order numerical strategy.

Impact Statement

Nonintrusive model order reduction (MOR) technique that can reduce models of generalized forms is desired in many industrial fields. Among available methods, MOR combined with artificial neural network (ANN) has great potential. However, most of the current ANN-based MOR methods ignore the fact that the target system is actually evaluated by an unknown numerical solver. Runge–Kutta neural network (RKNN) is a type of ANN that treats the time-series data as a result of numerical integration. This concept also makes RKNN more physics-informed, which can be a novel basis for further research. Therefore, investigating the capability of using this architecture for learning in the reduced space is meaningful.

1. Introduction

Physics-based simulation has been an integral part of product development and design as a cheaper alternative to physical prototyping. On the one hand, large computational resource enables using complex simulation models in the design phase. On the other hand, small form-factor hardware is also readily available, enabling “edge computing,” that is deploying compact computing devices in factories and smart buildings to enable data analysis. The existence of such hardware opens up the possibility of transferring the physics-based models from design phase into the operation phase. This is a key part of the digital twin vision (Hartmann et al., 2018; Rasheed et al., 2019). A digital twin running next to the device can enable novel industrial solutions such as model-based predictive maintenance or model-based process optimization.

However, bringing highly complex simulation models into the operation phase presents several challenges which are not present in the design phase. Firstly, the memory footprint of such models needs to be reduced to fit within the limited memory of edge devices, alongside the rest of the other potential data-analysis processes. Secondly, the models need to run at or faster than real-time on the limited hardware of edge devices. The methods which transform simulation models to comply with such requirements are known under the umbrella term of model order reduction (MOR; Antoulas, 2005; Hinze and Volkwein, 2005; Willcox and Peraire, 2002).

Most MOR techniques rely on mapping the high-dimensional state of the full order model (FOM) into a lower-dimensional space where the reduced order model (ROM) is to be solved. Within this contribution, we look into proper orthogonal decomposition (POD), one of the most widely used methods for finding such mapping (Willcox and Peraire, 2002; Hinze and Volkwein, 2005). POD is based on performing principal component analysis (PCA) on model trajectories generated by the original FOM, known as snapshots.

For linear models, knowing the dimension-reduction mapping and the equations which define the FOM is enough to obtain an ROM through projection. However, this cannot be done for complex models with nonlinearities. In this case, a model-reconstruction step is needed in order to reproduce the nonlinear behaviors in the reduced space. There are several methods perform this step, such as discrete empirical interpolation (Chaturantabut and Sorensen, 2010), operator inference (Peherstorfer and Willcox, 2016), and long-short-term-memory neural network (NN)s (Mohan and Gaitonde, 2018). Besides, recurrent NNs (Kosmatopoulos et al., 1995), have also been used for this purpose (Kani and Elsheikh, 2017; Wang et al., 2020). As a universal approximator, a standard multilayer perceptron (MLP) NN also can be used for this purpose. Sometimes, ROMs which are flexible with the time-stepping are required. In this case, networks such as RS-ResNet (Qin et al., 2019) can be employed. However, a disadvantage of this kind of networks is the requirement of training different networks for different time steps. Therefore, the flexibility is restricted by the number of the trained networks. Apart from these, we also would like to point out the numerical-integration-based NN models such as explicit Euler NN (EENN; (Pan and Duraisamy, 2018) and Runge–Kutta NN (RKNN; Wang and Lin, 1998) specialize in nonintrusively modeling the solution of ordinary differential equation (ODE) or partial differential equation (PDE). Moreover, these networks can efficiently learn the time information in the training data and be more flexible to the time stepping strategy. Therefore, their potential of reconstructing the ROMs is worth investigation.

The combination of POD for dimension reduction and machine learning represents a purely-data-driven MOR frameworkfor the complex problems (Agudelo et al., 2009; Ubbiali, 2017; Pawar et al., 2019). The main contribution of this work consists in exploring methodological variations to this framework. On the one hand, the effects of different approaches to generate the snapshot data for POD are investigated. On the other hand, the impact of using different NN architectures is explored. Although by some means the intrusive methods can be applied to reduce the test models investigated in this work, we will consider in a more generalized scope where we have very limited access to the FOM solver and a nonintrusive solution is necessary. Therefore, conventional intrusive methods are not studied in this work.

The paper is organized as follows. The introduction to POD and the modification for taking snapshots of the FOMs is given in Section 2. The principles and architectures of different networks are described in Section 3, numerical experiment evaluating the proposed MOR framework are given in Section 4, and their results are shown in Section 5. Finally, the conclusions follow in Section 6.

2. Proper Orthogonal Decomposition

In the last decades, there have been many efforts to develop different techniques in order to obtain compact low-dimensional representations of high-dimensional datasets. These representations, in general, encapsulate the most important information while discarding less important components. Some applications include image compression using PCA (Du and Fowler, 2007), data visualization using t-SNE (van der Maaten and Hinton, 2008) and structural description of data using diffusion maps (Coifman et al., 2005). Here, the focus will be on POD, which is a variant of PCA in the field of MOR.

POD can find a reduced basis $\boldsymbol{V}$ of arbitrary dimension that optimally represents (in a least square sense) the trajectories of the FOM used as snapshots. This basis can be used to project such snapshots into the low-dimensional space.

2.1. Problem statement

The full-dimensional problem that we intend to reduce in this work is an ordinary differential equation (ODE) typically originating from a spatial discretization from a multidimensional PDE. This ODE is frequently used as the governing equation for many engineering problems. The equation can be written as:

$$\dot{\boldsymbol{y}}(t) = \boldsymbol{f}(\boldsymbol{y}(t);\boldsymbol{\mu}(t)), \tag{1}$$

where $\boldsymbol{y} \in R^N$ is the state vector of the FOM, $t \in [t_0, t_{\text{end}}]$ is the time, and $\mu \in R^{n_\mu}$ is the vector of the system parameters. N is the number of variables in state vector will be called the size of the FOM and n_μ is the number of system parameters. The target of MOR is to find a reduced model, with size $N_r \ll N$, which can reproduce the solution of the FOM to a certain accuracy for a given set of system parameters $\boldsymbol{\mu}(t)$. This goal will be achieved by mapping the FOM into a reduced space with the help of a reduced basis $\boldsymbol{V}$. As briefly described before, the reduced basis is constructed from the snapshots of the FOM. Snapshots are nothing but a set of solutions, $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{N_s}\}$, to the FOM Equation (1) with corresponding system parameter configurations $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_{Ns}\}$. Here, N_s is the number of snapshots taken for the FOM. Often, these snapshots will present in the form of matrix called snapshot matrix $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{N_s}] \in R^{N \times (N_s \cdot k)}$, where k is the number of time steps in each simulation.

2.2. Taking snapshots: static-parameter sampling (SPS) and dynamic-parameter sampling (DPS)

Since the reduced basis $\boldsymbol{V}$ is constructed based on the snapshot matrix $\boldsymbol{Y}$ and later the snapshot will also be used to train the artificial NN (ANN), the quality of the snapshots is crucial to the performance of the whole framework. Here, in this paper, we propose a new sampling strategy, DPS, for taking snapshots to capture the dynamics of the FOM as much as possible.

The conventional way to define the system parameters $\boldsymbol{\mu}$ during each snapshot simulation is to choose *constant* values $\boldsymbol{\mu}(\boldsymbol{t}) = \boldsymbol{\mu}(\boldsymbol{t_0})$ using some sparse sampling technique such as *Sparse Grids*, *Latin Hypercube Sampling* (Helton and Davis, 2003), *Hammersley Sampling*, and *Halton Sampling* (Wong et al., 1997). Some strategies can select those constant parameter values smartly, among which the greedy sampling (Bui-Thanh, 2007; Haasdonk et al., 2011; Lappano et al., 2016) is probably the most well-known. Although greedy sampling techniques have been proven highly successful for projection-based, intrusive MOR methods, they are difficult to apply to the reduction methods used in this work. This is because we do not count with a priori error estimator and a posteriori error calculation would require the retraining of the NNs during the evaluation of the snapshots, which is unfeasible due to computational costs. Therefore, DPS is proposed in this work as the alternative snapshot collection strategy for non-intrusive MOR.

DPS, just as its name implies, we use *time-dependent* parameter values to construct the ith snapshot solution $\boldsymbol{y}_i$ so that it satisfies

$$\dot{\boldsymbol{y}}_i = \boldsymbol{f}(\boldsymbol{y}_i;\boldsymbol{\mu}_i(t)). \tag{2}$$

We would like to select the values for the $\boldsymbol{\mu}_i$ from a function space

$$\mathcal{F} = \{\mu : [t_0, t_{\text{end}}] \to \Omega\}, \tag{3}$$

where $[t_0, t_{\text{end}}] \subset R$ is the time span and $\Omega = [\mu_1^{min}, \mu_1^{max}] \times [\mu_2^{min}, \mu_2^{max}] \times \ldots \times [\mu_{n_\mu}^{min}, \mu_{n_\mu}^{max}]$ is a hypercube defined by the limits of the individual components of the the parameter vector. In principle, we could use

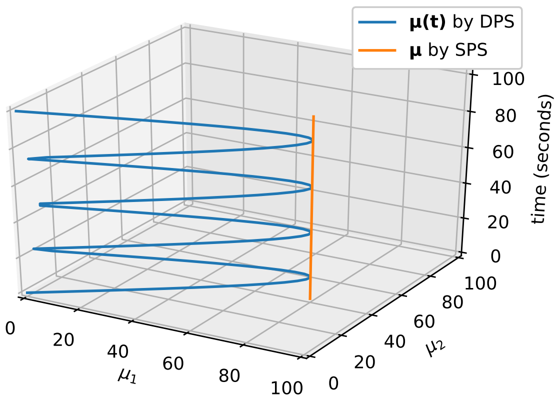


Figure 1. *An example showing the relation and difference between static-parameter sampling (SPS) and dynamic-parameter sampling (DPS). We assume the parameter space is designed as* $[0,100]\times[0,100]$. *Orange: the parameter configuration* $\boldsymbol{\mu}=[80,40]$ *is selected for running one snapshot simulation. Blue: the parameter configuration* $\boldsymbol{\mu}(t)=[80|sin(4\pi/100t)|,40|sin(4\pi/100t)|]$ *is selected for running one snapshot simulation.*

any finite subset of $\mathcal{F}$ to build our snapshots. In this work, we decided to use sinusoidal function for dynamic sampling as an example. These functions are constructed as follows:

1. Use any multivariate sampling, for example, Hammersley sequence is used in this work, to select N_s parameter vectors$\{\boldsymbol{\mu_1},\boldsymbol{\mu_2},\dots,\boldsymbol{\mu_{N_s}}\}$.
2. Select N_s randomly distributed values for the angular frequency $\{\omega_1,\dots,\omega_{N_s}\}\subset[0,\omega_{max}]$, where $\omega_{max}=4$, selected so that the functions have a maximum of 4 oscillations within the time frame.
3. Define the parameter functions as

$$\boldsymbol{\mu_i}(t)=\boldsymbol{\mu}_i^{amp}|sin(\omega t)|+\boldsymbol{\mu_i}^{min}, \tag{4}$$

where $\boldsymbol{\mu}_i^{amp}=\boldsymbol{\mu_i}-\boldsymbol{\mu}_{min}$ and $\boldsymbol{\mu}_{min}=\left[\mu_1^{min},\mu_2^{min},\dots,\mu_{n_\mu}^{min}\right]$.

A graphical representation of representative function is shown in Figure 1.

The solutions of Equation (2) using parameters defined by Equation (4) are the snapshots we use for DPS. The snapshots contain the dynamic response of the system with certain parameter configurations. And more dynamic modes are included in the snapshots, more essential information about the system is available to the reduction. Moreover, since the input–output response of the system will be fed into the NN in Section 3, the diversity of the snapshots will strongly influence the training procedure. In this aspect, DPS can provide us with more diverse observation to the system's response.

We must clarify that the procedure used to define the input parameters functions is arbitrary. The sinusoidal range and frequency of the sinusoidal function should: (a) cover the whole relevant value range of the parameters and (b) incorporate time dependent effects. The limit in the maximum frequency chosen relatively small because of the nature of the problems used as test cases. It is possible to consider the use of other functions instead of sinusoidal functions. As an example, Qin et al. (2021) have defined time-dependent input functions using low-order polynomials. We favor sinusoidal functions, because we can more easily ensure that their values fall within the limits we define for the individual parameters. We defer the exploration of alternatives to future research.

2.3. Singular value decomposition

With the snapshots obtained in the Section 2.2, the goal is to find an appropriate reduced basis $\{\boldsymbol{v_i}\}_{i=1}^{N_r}$ which minimizes the approximation error (Chaturantabut and Sorensen, 2010):

$$\epsilon_{\text{approx}} = \sum_{j=1}^{N_s} \left\| \boldsymbol{y}_j - \sum_{i+1}^{N_r} \left(\boldsymbol{y}_j^T \boldsymbol{v}_i \right) \boldsymbol{v}_i \right\|_2^2, \tag{5}$$

where $\boldsymbol{y_i}$ stands for $\boldsymbol{y}(\boldsymbol{\mu_i})$.

The minimization of ϵ_{approx} in Equation (5) can be solved by singular value decomposition of the snapshot matrix $\boldsymbol{Y} = [\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_{N_s}}] \in R^{N \times (N_s \cdot k)}$:

$$\boldsymbol{Y} = \boldsymbol{V} \Sigma \boldsymbol{W}^T. \tag{6}$$

If the rank of the matrix $\boldsymbol{Y}$ is k, then the matrix $\boldsymbol{V} \in R^{N \times k}$ consists of k column vectors $\{\boldsymbol{v}_i, i = 1, 2, \ldots, k\}$, and matrix $\boldsymbol{\Sigma}$ is a diagonal matrix $diag(\sigma_1, \sigma_2, \ldots, \sigma_k)$. σ_i is called ith singular value corresponding to ith singular vector $\boldsymbol{v_i}$.

Essentially, each singular vector $\boldsymbol{v}_i$ represents a dynamic mode of the system. And the corresponding singular value σ_i of singular vector $\boldsymbol{v}_i$ can be seen as the "weight" of the dynamic mode $\boldsymbol{v}_i$. Since the greater the weight, the more important the dynamic mode is, N_r singular vectors corresponding to the greatest N_r singular values will be used to construct the reduced basis $\boldsymbol{V}_r = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{N_r}] \in R^{N \times N_r}$. Theoretically, the reduced model will have better quality if more dynamic modes are retained in $\boldsymbol{V}$, but this will also increase the size of the reduced model. And since some high frequency noise with small singular value might also be observed in snapshots and later included in the reduced model, the reduced dimension must be selected carefully. For research regarding this topic, we refer to Josse and Husson (2012) and Valle et al. (1999), and we will not further study it in this work.

The high-dimensional Equation (1) can be projected into the reduced space using the reduced basis:

$$\boldsymbol{V}^T \boldsymbol{V} \dot{\boldsymbol{y}}_{\boldsymbol{r}}(t) = \boldsymbol{V}^T \boldsymbol{f}(\boldsymbol{V}\boldsymbol{y}_{\boldsymbol{r}}; \boldsymbol{\mu}), \tag{7}$$

where $\boldsymbol{y_r}$ is the reduced state vector. And if the orthonormality of the column vectors in $\boldsymbol{V}$ is considered, the Equation (7) can be simplified to:

$$\dot{\boldsymbol{y}}_{\boldsymbol{r}}(t) = \boldsymbol{V}^T \boldsymbol{f}(\boldsymbol{V}\boldsymbol{y}_{\boldsymbol{r}}; \boldsymbol{\mu}). \tag{8}$$

In Equation (8), if $\boldsymbol{f}(\cdot)$ is a linear function of $\boldsymbol{y} \approx \boldsymbol{V}\boldsymbol{y_r}$, then further reduction can be applied and leads to:

$$\dot{\boldsymbol{y}}_{\boldsymbol{r}}(t) = \boldsymbol{f}_{\boldsymbol{r}}(\boldsymbol{y}_{\boldsymbol{r}}; \boldsymbol{\mu}). \tag{9}$$

However, if we consider a more generalized form of $\boldsymbol{f}(\cdot)$, that is the function can be either linear or nonlinear to $\boldsymbol{y}$, the evaluation of $\boldsymbol{f}(\cdot)$ still requires lifting $\boldsymbol{y_r}$ to $\boldsymbol{V}\boldsymbol{y_r}$. This lifting and evaluation is performed during the online phase and can significantly reduce the efficiency of the ROM. A solution to this problem can be using an approximator (e.g., NN) to construct a new function satisfying $\widehat{\boldsymbol{f}}(\boldsymbol{y_r}; \boldsymbol{\mu}) \approx \boldsymbol{V}^T \boldsymbol{f}(\boldsymbol{V}\boldsymbol{y_r}; \boldsymbol{\mu})$.

3. Prediction by NN

After finding the projection into the lower-dimensional space, we need a way to reproduce the projected dynamics which is governed by Equation (1) in the full-order space. In this work, we consider three NN architectures: MLP, EENN, and RKNN.

In all cases the training data uses the snapshots projected into the reduced space.

$$\begin{aligned} \boldsymbol{Y}_r &= \boldsymbol{V}^{\boldsymbol{T}} \boldsymbol{Y} \\ &= \boldsymbol{V}^{\boldsymbol{T}} [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N_s}] \\ &= [\mathbf{y}_{r,1}, \mathbf{y}_{r,2}, \ldots, \mathbf{y}_{r,N_s}], \end{aligned} \tag{10}$$

where

$$\mathbf{y}_{r,i} = [\mathbf{y}_{r,i}(t_1), \ldots, \mathbf{y}_{r,i}(t_k)] \tag{11}$$

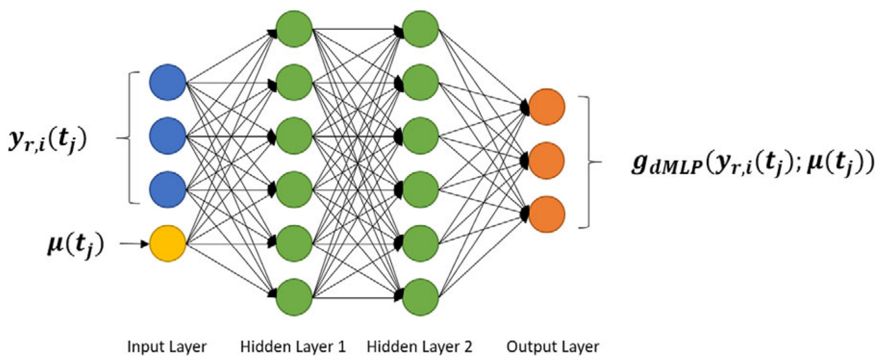


Figure 2. *Picture of the structure of multilayer perceptron (MLP). MLPs approximate the new state $\boldsymbol{y}_{r,i}(t_{j+1})$ based on the given input $\boldsymbol{y}_{r,i}(t_j)$ and $\boldsymbol{\mu}(t_j)$.*

is the projected trajectories corresponding to the parameter $\mu_i(t)$ and k is the number of time steps used for each snapshot simulation. If we denote the step size with τ, we will have

$$t_j = t_0 + \tau j \text{ for } 0 \le j \le k. \tag{12}$$

We also note that $\mathbf{Y}_r \in \mathbb{R}^{N_r \times (N_s \cdot k)}$.

All the types of networks in this paper use the rectified linear unit (Glorot et al., 2011) as the activation functions for enhancing the capability to approximating generalized nonlinearity. For updating the parameters of the networks, the backpropagation (Rumelhart et al., 1986) is the method applied in this work. For doing this, the Adam optimizer (Kingma and Ba, 2014) is used and L2 regularization is deployed to prevent overfitting.

3.1. Multilayer perceptron

If the time step in the training data is the same as the desired time step in validation, the most straightforward strategy of predicting the state in the future is to map all current information, that is state vector and input parameters, to the future state vector. This is exactly how MLP network learns to predict the evolution of the system. Its concept of learning the relation between each pair of neighboring state vectors is shown in Figure 2.

$$\boldsymbol{y}_{r,i}(t_{j+1}) \approx \boldsymbol{g}_{\text{dMLP}}(\mathbf{y}_{r,i}(t_j), \boldsymbol{\mu}(t_j)). \tag{13}$$

Algorithm 1 Training of MLP network

```
Require: $\boldsymbol{y}_{r,i}(t_j), \boldsymbol{\mu}(t_j), \boldsymbol{y}_{r,i}(t_{j+1})$
1: while $loss > loss_{tol}$ do
2:     $\widehat{\boldsymbol{y}}_{r,i}(t_{j+1}) = \boldsymbol{g}_{\text{dMLP}}([\boldsymbol{y}_{r,i}(t_j), \boldsymbol{\mu}(t_j)])$
3:     $loss = MSE(\widehat{\boldsymbol{y}}_{r,i}(t_{j+1}), \boldsymbol{y}_{r,i}(t_{j+1}))$
4:     Use backpropagation to update the weights and biases of the network $\boldsymbol{g}_{\text{dMLP}}$
5: end while
```

In Algorithm 1, $\boldsymbol{y}_{r,i}(t_j)$ and $\boldsymbol{y}_{r,i}(t_{j+1})$ are the state vectors stored in the snapshots, and $\boldsymbol{\mu}(t_j)$ is the parameter vector at the corresponding time instance. The operation $[A,B]$ means to concatenate two arrays. Without loss of generality, we assume both $\boldsymbol{y}_{r,i}(t_j)$ and $\boldsymbol{\mu}(t_j)$ are column vectors, then this operation will vertically stack them in sequence. The loss function used in this paper is mean-square error (MSE), it computes the error between the approximation and the target by:

$$MSE(\boldsymbol{L}_1, \boldsymbol{L}_2) = \frac{1}{n}\sum_{i=0}^{n}(\boldsymbol{L}_1[i] - \boldsymbol{L}_2[i])^2 \tag{14}$$

here n is the number of elements in both arrays, and i in the square brackets means ith element of the array. The same notation and loss function is also used for the networks introduced in the next sections.

3.2. *Explicit Euler neural network*

As an alternative to learning the mapping between states of two consecutive time steps, it is possible to learn the approximation $\widehat{\boldsymbol{f}}(\cdot)$ to the R.H.S. of the reduced governing equations (Equation (8)). In this case, the inputs to the NN are again the current state of the system $\boldsymbol{y}_{r,i}(t_j)$ and $\boldsymbol{\mu}(\boldsymbol{y}_j)$, but the new state is now calculated as

$$\boldsymbol{y}_{r,i}(t_{j+1}) = \boldsymbol{y}_{r,i}(t_j) + \tau \boldsymbol{g}_{\text{eeMLP}}(t_j, \mu), \tag{15}$$

where $\boldsymbol{g}_{\text{eeMLP}}(t_j, \mu)$ is an MLP used as the approximator to the R.H.S.

We denote a network trained in this way an EENN, as the approximation of the R.H.S. Equation (15) corresponds to the explicit Euler (EE) integration scheme. In the test cases where the time steps are kept constant, EENN and MLP are expected to perform very similarly. However, learning the dynamics with such methods can present advantages when flexibility in time steps size is necessary. Therefore, in this work we will compare MLP and RKNN with constant-time-step tests and compare EENN and RKNN with variant-time-step tests.

From Equation (15), we know that an EENN is essentially a variant of residual network (He et al., 2016) which learns the increment between two system states instead of learning to map the new state directly from the old state. This leads to a potential advantage of EENNs that we can use deeper network for approximating more complex nonlinearity.

The algorithm for training such an EENN is provided in Algorithm 2 and the sketch of the network structure is given in Figure 3.

Algorithm 2 Training of EENN

```
Require: $\boldsymbol{y}_{r,i}(t_j)$, $\boldsymbol{\mu}(t_j)$, $\boldsymbol{y}_{r,i}(t_{j+1})$
 1: while $loss > loss_{tol}$ do
 2:     $\widehat{\boldsymbol{y}}_{r,i}(t_{j+1}) = \boldsymbol{y}_{r,i}(t_j) + \tau \boldsymbol{g}_{\text{eeMLP}}([\boldsymbol{y}_{r,i}(t_j), \boldsymbol{\mu}(t_j)])$
 3:     $loss = MSE(\widehat{\boldsymbol{y}}_{r,i}(t_{j+1}), \boldsymbol{y}_{r,i}(t_{j+1}))$
 4:     Use backpropagation to update the weights and biases of the network $\boldsymbol{g}_{\text{eeMLP}}$
 5: end while
```

3.3. *Runge–Kutta neural network*

We also consider in this work RKNN, which embed NNs into higher order numerical integration schemes. In this work, we only focus on explicit fourth-order Runge–Kutta (RK) as it is the most widely used version used in engineering problems, however the approach can be applied to any order. The fourth-order RK integration can be represented as follows:

$$\boldsymbol{y}_{r,i}(t_{j+1}) = \boldsymbol{y}_{r,i}(t_j) + \frac{1}{6}(\boldsymbol{h}_1 + 2\boldsymbol{h}_2 + 2\boldsymbol{h}_3 + \boldsymbol{h}_4), \tag{16}$$

where:

$$\boldsymbol{h}_1 = \tau \boldsymbol{f}(\boldsymbol{y}_{r,i}(t_j); \boldsymbol{\mu}(t_j)), \qquad \boldsymbol{h}_2 = \tau \boldsymbol{f}\left(\boldsymbol{y}_{r,i}(t_j) + \frac{\boldsymbol{h}_1}{2}; \boldsymbol{\mu}(t_j)\right)$$
$$\boldsymbol{h}_3 = \tau \boldsymbol{f}\left(\boldsymbol{y}_{r,i}(t_j) + \frac{\boldsymbol{h}_2}{2}; \boldsymbol{\mu}(t_j)\right), \quad \boldsymbol{h}_4 = \tau \boldsymbol{f}(\boldsymbol{y}_{r,i}(t_j) + \boldsymbol{h}_3; \boldsymbol{\mu}(t_j)).$$

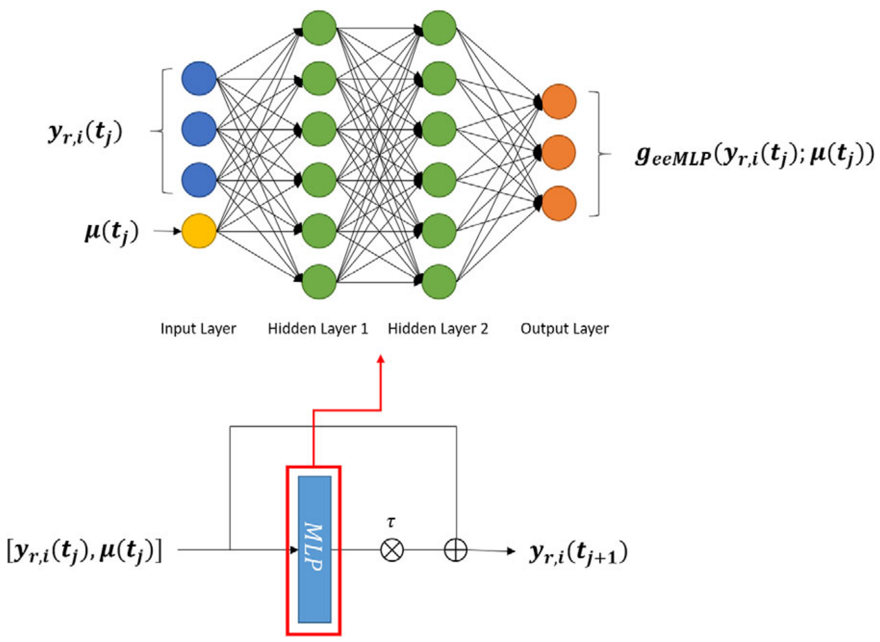


Figure 3. *Picture of an explicit Euler neural network (EENN). EENN uses a multilayer perceptron (MLP) to approximate the R.H.S. of Equation (8). The output of the MLP is assembled as described in Equation (15).*

As we can see, the scheme requires the ability to evaluate the R.H.S. $f(\boldsymbol{y},\mu)$ four times per time step. The concept of this approach is based on using a MLP NN to approximate this function

$$\boldsymbol{f}(\boldsymbol{y}_r;\boldsymbol{\mu}) \approx \boldsymbol{g}_{\text{rkMLP}}(\boldsymbol{y}_r;\boldsymbol{\mu}). \tag{17}$$

Then this MLP can be integrated into the RK scheme as subnetwork to obtain an integrator

$$\boldsymbol{y}_{r,i}(t_{j+1}) = g_{\text{RKNN}}(\boldsymbol{y}_{r,i}(t_j);\boldsymbol{\mu}(t_j)) = \boldsymbol{y}_{r,i}(t_j) + \frac{1}{6}\left(\boldsymbol{h}_1^{\text{RKNN}} + 2\boldsymbol{h}_2^{\text{RKNN}} + 2\boldsymbol{h}_3^{\text{RKNN}} + \boldsymbol{h}_4^{\text{RKNN}}\right), \tag{18}$$

where

$$\boldsymbol{h}_1^{\text{RKNN}} = \tau\boldsymbol{g}_{\text{rkMLP}}(\boldsymbol{y}_{r,i}(t_j);\boldsymbol{\mu}(t_j)), \qquad \boldsymbol{h}_2^{\text{RKNN}} = \tau\boldsymbol{g}_{\text{rkMLP}}\left(\boldsymbol{y}_{r,i}(t_j) + \frac{\boldsymbol{h}_1^{\text{RKNN}}}{2};\boldsymbol{\mu}(t_j)\right)$$

$$\boldsymbol{h}_3^{\text{RKNN}} = \tau\boldsymbol{g}_{\text{rkMLP}}\left(\boldsymbol{y}_{r,i}(t_j) + \frac{\boldsymbol{h}_2^{\text{RKNN}}}{2};\boldsymbol{\mu}(t_j)\right), \qquad \boldsymbol{h}_4^{\text{RKNN}} = \tau\boldsymbol{g}_{\text{rkMLP}}(\boldsymbol{y}_{r,i}(t_j) + \boldsymbol{h}_3^{\text{RKNN}};\boldsymbol{\mu}(t_j)).$$

In Figure 4, we present a schematic on how the MLP subnetwork is embedded into the larger RKNN based on Equation (18). It is worth noticing that RKNNs also use a residual network structure.

Let us take the computation of $\boldsymbol{h}_1^{\text{RKNN}}$ in Equation (18) as an example. We consider an MLP as subnetwork with one input layer, two hidden layers, and one output layer and all fully interconnected. The input layer receives the previous state vector $\boldsymbol{y}_{r,i}(t_j)$ and the parameter vector $\boldsymbol{\mu}(t_j)$, that is the number of input neurons is set according to the number of components in the reduced space and the number of parameters. That is, the amount of neurons will depend exclusively on the model and it might change radically for different models. The output of this MLP becomes the approximation for $\boldsymbol{h}_1$. The way subnetwork computing $\boldsymbol{h}_2^{\text{RKNN}}, \boldsymbol{h}_3^{\text{RKNN}}$, and $\boldsymbol{h}_4^{\text{RKNN}}$ are similar and the only difference is the reduced state vector $\boldsymbol{y}_{r,i}(t_j)$ in the input will be replaced by $\boldsymbol{y}_{r,i}(t_j) + c_i\boldsymbol{h}_i^{\text{RKNN}}$,

Algorithm 3 Training of RKNN

```
Require: $\boldsymbol{y}_{r,i}(t_i), \boldsymbol{\mu}(t_j), \boldsymbol{y}_{r,i}(t_{i+1}), \tau$
 1: while loss > loss_{tol} do
 2:     $\boldsymbol{h}_1^{\text{RKNN}} = \tau \boldsymbol{g}_{\text{rkMLP}}\left(\left[\boldsymbol{y}_{r,i}(t_j), \boldsymbol{\mu}(t_j)\right]\right)$
 3:     $\boldsymbol{h}_2^{\text{RKNN}} = \tau \boldsymbol{g}_{\text{rkMLP}}\left(\left[\boldsymbol{y}_{r,i}(t_j) + \frac{\boldsymbol{h}_1^{\text{RKNN}}}{2}, \boldsymbol{\mu}(t_j)\right]\right)$
 4:     $\boldsymbol{h}_3^{\text{RKNN}} = \tau \boldsymbol{g}_{\text{rkMLP}}\left(\left[\boldsymbol{y}_{r,i}(t_j) + \frac{\boldsymbol{h}_2^{\text{RKNN}}}{2}, \boldsymbol{\mu}(t_j)\right]\right)$
 5:     $\boldsymbol{h}_4^{\text{RKNN}} = \tau \boldsymbol{g}_{\text{rkMLP}}\left(\left[\boldsymbol{y}_{r,i}(t_j) + \boldsymbol{h}_3^{\text{RKNN}}, \boldsymbol{\mu}(t_j)\right]\right)$
 6:     $\widehat{\boldsymbol{y}}_{r,i}(t_{j+1}) = \boldsymbol{y}_{r,i}(t_j) + \frac{1}{6}\left(\boldsymbol{h}_1^{\text{RKNN}} + 2\boldsymbol{h}_2^{\text{RKNN}} + 2\boldsymbol{h}_3^{\text{RKNN}} + \boldsymbol{h}_4^{\text{RKNN}}\right)$
 7:     $loss = MSE\left(\widehat{\boldsymbol{y}}_{r,i}(t_{i+1}), \boldsymbol{y}_{r,i}(t_{i+1})\right)$
 8:     Use backpropagation to update the weights and biases of the network $\boldsymbol{g}_{\text{rkMLP}}$
 9: end while
```

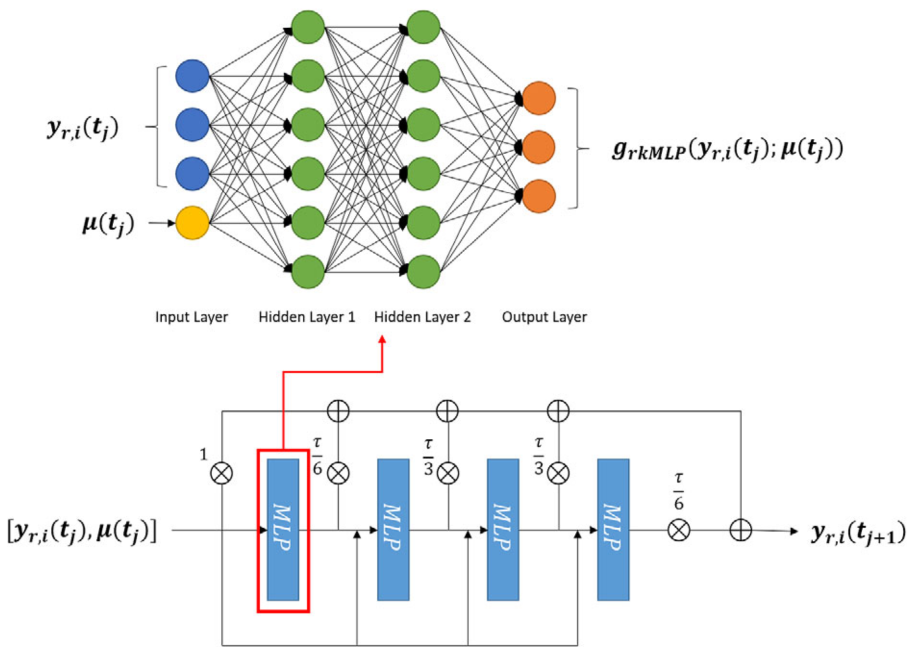
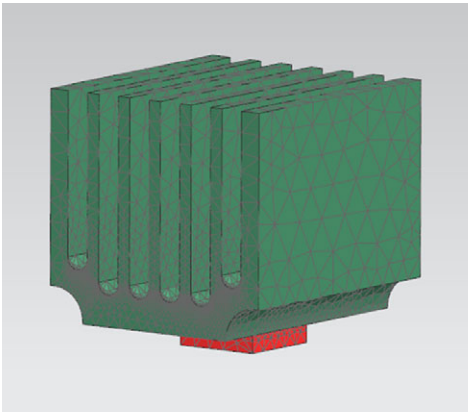

$\boldsymbol{y}_{r,i}(\boldsymbol{t}_j)$
$\boldsymbol{\mu}(\boldsymbol{t}_j)$
$\boldsymbol{g}_{\boldsymbol{rkMLP}}(\boldsymbol{y}_{r,i}(\boldsymbol{t}_j); \boldsymbol{\mu}(\boldsymbol{t}_j))$
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer
1
$\frac{\tau}{6}$
$\frac{\tau}{3}$
$\frac{\tau}{3}$
$\frac{\tau}{6}$
MLP
MLP
MLP
MLP
$[\boldsymbol{y}_{r,i}(\boldsymbol{t}_j), \boldsymbol{\mu}(\boldsymbol{t}_j)]$
$\boldsymbol{y}_{r,i}(\boldsymbol{t}_{j+1})$

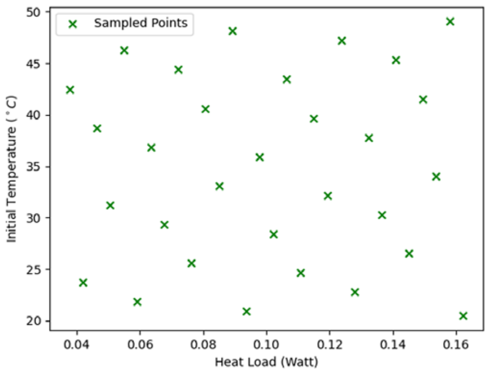
Figure 4. *Picture of a Runge–Kutta neural network (RKNN). An RKNN has a multilayer perceptron (MLP) as the core network which approximates the R.H.S. of Equation (8). The output of the network is assembled as described in Equation (16).*

where c_i is the coefficient in Equation (16) and $\boldsymbol{h}_i^{\text{RKNN}}$ is computed in previous iteration. The parameters of the subnetwork are updated using backpropagation. Also it is worth mentioning that although there will be multiple computation for the intermediate stages of the RK scheme, only one MLP subnetwork is needed for doing that. Therefore, the size of an RKNN is not affected by the order of the RK scheme.

Similar to explicit integrators, implicit ones can be used (Rico-Martinez and Kevrekidis, 1993). Due to their recurrent nature, or the dependence of predictions on themselves, the architecture of this type of NN will have recurrent connections and offer an alternative to the approach taken here.



(a) Picture of the heat sink model



(b) Sampled parameters

***Figure 5.** Test case: heat sink model.*

4. Numerical Examples

In this section, the three numerical examples used for testing are described. The first one is a computer heat sink, which is governed by the thermal equation without any nonlinear contribution. The second one is a gap-radiation model whose physics equation is the thermal equation including fourth-order nonlinear radiation terms. And the last example is a thermal-fluid model simulating a heat exchanger, which includes the effects of fluid flow.

All test cases are all firstly reduced by POD method and then ROMs are evaluated by both MLP and RKNN. And to validate the ROMs, each example will be assigned two test cases, one is a constant-load test and the other one is a dynamic-load test.

The numerical experiments are performed on a computer with Intel Xeon E5-2640 CPU (6 cores and 2.50-3.00 GHz) and 48 GB memory. The snapshot simulations and reference solutions are provided by thermal/flow multi-physics module in simulation software Siemens Simcenter NX 12 (release 2020.1, version 1915).[1] The open-source machine learning library Pytorch (Paszke et al., 2017) is used to realize the training of the NNs. Some special treatment to improve the network's training is provided in Section 3.1, additionally, learning rate decay is employed to select the optimal learning rate adaptively. We also use early stopping (Prechelt, 1998) to prevent the networks from overfitting.

4.1. Heat sink model

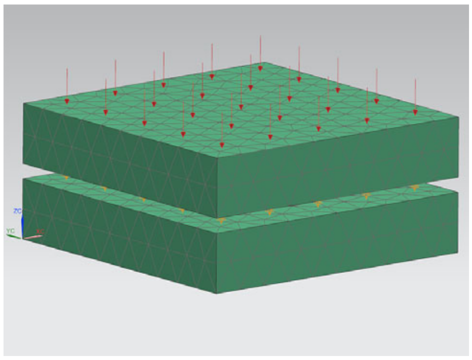
The first FEM model simulates a chip cooled by the attached heat sink. The chip will be the heat source of the system and heat flux travels from the chip to the heat sink then released into the environment through fins. Since the material used in the model is not temperature-dependent, the system is governed by the linear heat transfer equation as Equation (19).

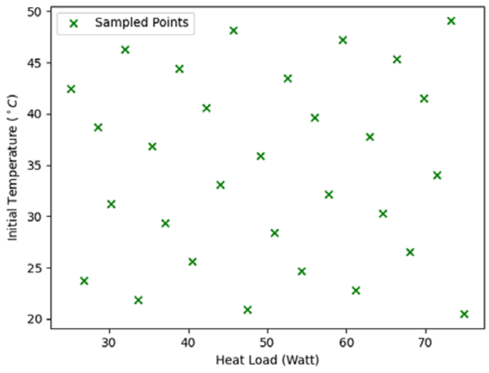
$$\boldsymbol{C_p}\dot{\boldsymbol{T}} = \boldsymbol{KT} + \boldsymbol{Bu}, \tag{19}$$

where $\boldsymbol{C_p}$ is the thermal capacity matrix, $\boldsymbol{K}$ is the thermal conductivity matrix, $\boldsymbol{B}$ is the generalized load matrix, $\boldsymbol{T}$ is the state vector of temperature, and $\boldsymbol{u}$ is the input heat load vector.

The meshed model is sketched in Figure 5a. The model consists of two parts, the red part is a chip made from copper whose thermal capacity is 385 J kg^{-1} K^{-1} and thermal conductivity is 387 W m^{-1} K^{-1}. The

[1] https://www.plm.automation.siemens.com/global/en/products/simcenter/


(a) Picture of the gap-radiation model



(b) Sampled parameters

Figure 6. *Test case: gap-radiation model.*

green part represents the attached heat sink made from aluminum. The relevant material properties are 963 J kg^{-1} K^{-1} thermal capacity and 151 W m^{-1} K^{-1} thermal conductivity.

In the designed space, there are two variable parameters: initial temperature T_0 and input heat load u. The range of initial temperature is from 20°C to 50°C. The power of the volumetric heat source in the chip has range from 0.05 to 0.15 W/mm^3. Parameter coordinates sampled in parameter space are shown in Figure 5b.

Both test cases have uniform initial temperature of 20°C. The volumetric heat source used by constant-load test has fixed power 0.1 W/mm^3 and the one used by dynamic-load test has power as function of time $u(t) = 0.15 - 0.1\frac{t}{500}$ W/mm^3.

4.2. Gap-radiation model

The model presented here is a thermal model including radiative coupling. This model can be thought of as a proxy for other radiation-dominated heat transfer models such as those occurring in aerospspace, energy and manufacturing. The discrete governing equation is as Equation (20).

$$\boldsymbol{C_p}\dot{\boldsymbol{T}} = \boldsymbol{KT} + \boldsymbol{RT}^4 + \boldsymbol{Bu} \tag{20}$$

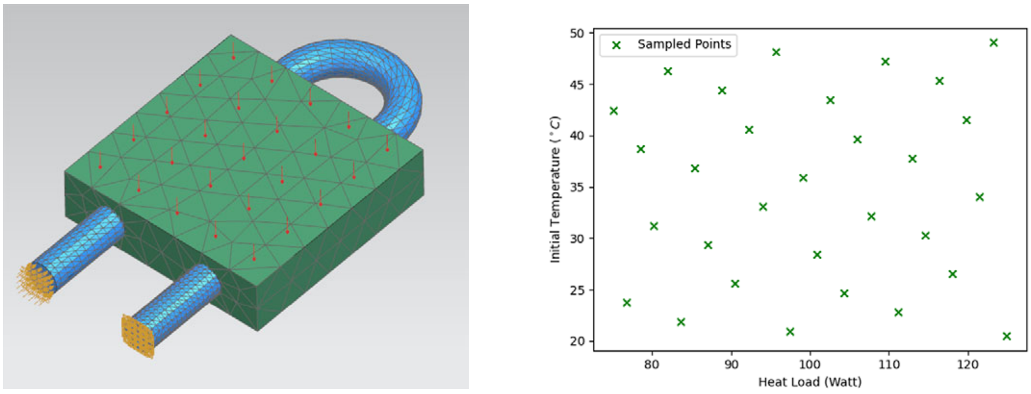
in addition to Equation (19), $\boldsymbol{R}$ is the radiation matrix and the operation $(\cdot)^4$ means to apply fourth-order power element-wise to the temperature vector, and not matrix multiplication.

The three-dimensional (3D) model is sketched in Figure 6a. As shown, the upper plate is heated by thermal flow. When the temperature equilibrium between the two plates is broken due to an increment of the temperature of the upper plate, radiative flux due to the temperature difference between two surfaces of the gap takes place.

Both plates are made of steel, whose thermal capacity is 434 J kg^{-1} K^{-1} and thermal conductivity is 14 W m^{-1} K^{-1}. The initial temperature T_0 and the applied heat load u are the variable parameters. The time scale of the whole simulated process is from 0 to 3,600 s.

Assuming operation condition where the input heat load of the system has the lower limitation $u_{min} =$ 40W and upper limitation $u_{max} = 60$W and the initial temperature has the lower limitation $T_{0,\min} = 20$°C and upper limitation $T_{0,\max} = 300$°C is investigated. So the parameter space to take snapshots is defined as [20°C, 300°C] × [40W, 60W]. And the parameter configurations used for snapshots are given in Figure 6b.

The test scenarios include a constant-load test and a dynamic-load test (Figure 7). The constant-load test has fixed load magnitude of 50 W, while the dynamic-load test has a time-dependent load magnitude $u(t) = 60 - 20\frac{(t-1{,}800)^2}{1{,}800^2}$ W.



(a) Picture of the heat exchanger model (b) Sampled parameters

Figure 7. *Test case: heat exchanger model.*

4.3. Heat exchanger model

In this example, we model the temperature profile of a solid that is in contact with a fluid channels to cool it down. The flow-field is governed by the Navier–Stokes (N–S) equations written in integral form (Kassab et al., 2003):

$$\int_{\Omega} \frac{\partial W}{\partial t} d\Omega + \int_{\Gamma} (F - T) \cdot n d\Gamma = \int_{\Omega} S d\Omega, \tag{21}$$

where Ω denotes the volume, Γ denotes the surface bounded by the volume Ω, and n is the outward-drawn normal. The conserved variables are contained in $W = (\rho, \rho u, \rho v, \rho w, \rho e, \rho k, \rho \omega)$, where, $\rho, u, v, w, e, k, \omega$ are the density, the velocity in x-, y-, and z-directions, and the specific total energy. F and T are convective and diffusive fluxes, respectively, S is a vector containing all terms arising from the use of a noninertial reference frame as well as in the production and dissipation of turbulent quantities.

The governing equation for heat-conduction field in the solid is:

$$\nabla \cdot [k_s(T_s) \nabla T_s] = 0 \tag{22}$$

here T_s denotes the temperature of the solid, and k_s is the thermal conductivity of the solid material.

Finally, the equations should be satisfied on the interface between fluid and solid are:

$$T_f = T_s k_f \frac{\partial T_f}{\partial n} = -k_s \frac{\partial T_s}{\partial n} \tag{23}$$

here T_f is the temperature computed from N–S solution Equation (21) and k_f is thermal conductivity of fluid.

The liquid flowing in the cooling tube is water whose thermal capacity is 4,187 J kg^{-1} K^{-1} and thermal conductivity is 0.6 W m^{-1} K^{-1}. The fluid field has an inlet boundary condition which equals to 1 m/s. The solid piece is made of aluminum whose thermal capacity is 963 J kg^{-1} K^{-1} and thermal conductivity is 151 W m^{-1} K^{-1}.

The initial temperature in the designed parameter space is from 20°C to 50°C and the heat load applied on the solid component varies from 80 to 120 W, as shown in Figure A1. Therefore, we will use a constant heat load whose magnitude is 100 W and a variant heat load whose magnitude is a function of time $u(t) = 120 - 0.8t$ W to validate the ROM, respectively.

5. Results

In this section, the results of the numerical tests in Section 4 are presented. The comparison will be made along two dimension: the first dimension is sampling method, that is SPS versus DPS. The other

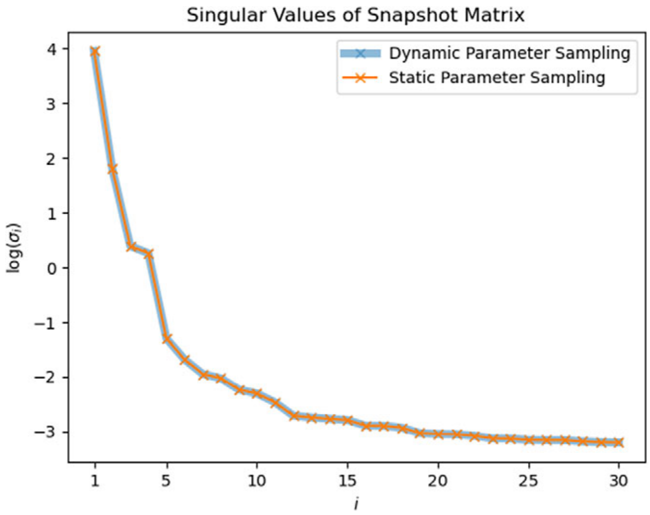


Figure 8. *Heat sink model: singular values of static-parameter sampling (SPS) snapshots and dynamic-parameter sampling (DPS) snapshots.*

dimension is architecture of NN, that is among MLP, EENN, and RKNN. Some qualitative and quantitative conclusion will be drawn based on comparison.

In Section 5.1, we will investigate how sampling strategy influences construction of reduced basis. And in Sections 5.2.1 and 5.2.2, we will focus on how dataset and architecture influences the learning quality respectively. For the investigation of different architectures, we focus on using MLPs and RKNNs to learn and predict with constant time step. In Section 5.2.3, we further study the capability of EENNs and RKNNs to learn from the snapshots sampled on the coarse time grids and to predict on the fine time grids.

5.1. POD reduction

With SPS and DPS respectively, 30 snapshots are taken for each system with 30 different heat-load magnitudes and initial temperatures. Each snapshot is a transient analysis from $t_0 = 0$ to t_{end} with 100 time points solved by NX 12.

The first 30 singular values found from SPS snapshots and DPS snapshots are shown in Figures 8–10 for each test case respectively. In the heat sink test model, the curves of σ from calculated from different sampling strategies seem to exactly overlap on each other. And in Figures 9 and 10, only slight difference can be found between two curves.

However, this does not mean the reduced basis constructed by different dataset has the same quality. To quantitatively measure the quality of reduced basis, here we perform re-projection test to calculate the error generated by each reduced basis. In the test, the reference trajectory is firstly projected into then reduced space then back-projected into the full space. This process can be done using $\widehat{\boldsymbol{Y}}_{ref} = \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{Y}_{ref}$. Depending on re-projected trajectory, we can calculate the relative re-projection error using $E_{POD} = \frac{1}{N_s \cdot k} \frac{\left|\boldsymbol{Y}_{ref} - \widehat{\boldsymbol{Y}}_{ref}\right|_2}{\left|\boldsymbol{Y}_{ref}\right|_2}$, where N_s is number of snapshot simulations and k is the number of time points in each simulation.

The POD errors with different sizes of reduced basis are shown in Figures A1 and A2. Similar to summarized above, in most situations, POD error curves generated by different sampling strategies can converge to similar values when the size of ROM is large enough. Nevertheless, in Figure Ab, the re-projection error of SPS-ROM converges to a value greater than DPS-ROM’s. This means the reduced basis calculated by DPS snapshots has higher quality (Figure A3).

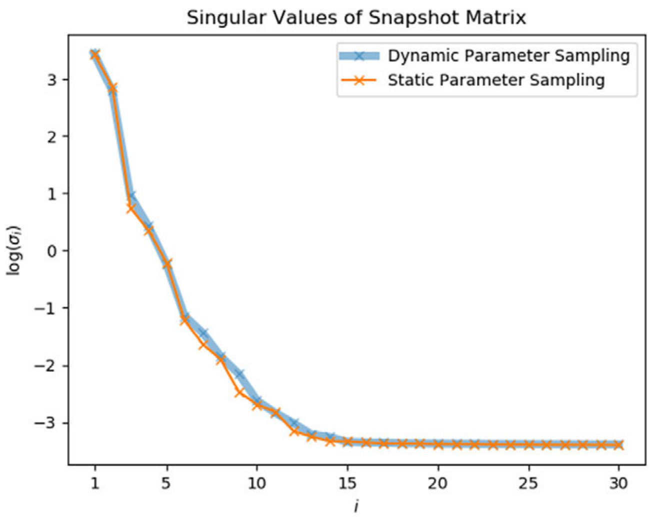


Figure 9. *Gap-radiation model: singular values of static-parameter sampling (SPS) snapshots and dynamic-parameter sampling (DPS) snapshots.*

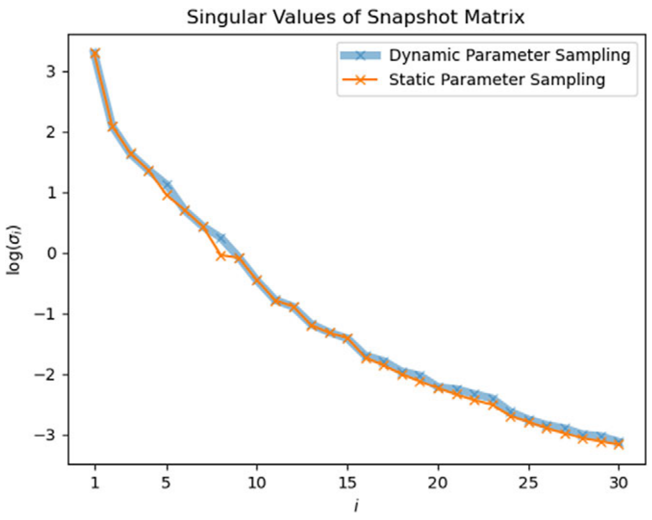


Figure 10. *Heat exchanger model: singular values of static-parameter sampling (SPS) snapshots and dynamic-parameter sampling (DPS) snapshots.*

It is observed that, SPS-ROM can have smaller projection error in some cases. This suggests that hybrid datasets (including static and dynamic parameters) may be needed to improve this aspect even further. We propose to study this possibility in future works.

5.2. ANN-based ROM

The quality of the ROM is affected by the reduced basis constructed by POD and the NN trained in the reduced space. In Section 5.1, by comparing singular values calculated from snapshot matrices sampled in different approaches, we did not see different sampling strategies have huge impact on constructing the POD subspace. But they can still make difference on training the NNs. This will be investigated in

Section 5.2.1. In Section 5.2.2, we compare MLP and RKNN with constant-step-size tests. In Section 5.2.3, we compare EENN with RKNN with variant-step-size tests.

5.2.1. Static-parameter sampling versus dynamic-parameter sampling

In Section 2.2, we introduced two approaches for taking snapshots. In conventional SPS, the parameter configuration assigned to the system is constant during the simulation time, that is $\boldsymbol{\mu}(t) = \boldsymbol{\mu}_0$. To include more parameter diversity in training data, we designed new sampling strategy named DPS whose parameter configuration is function of time. Datasets sampled by different strategies influences not only the quality of reduced basis but also the training of NN. In this section, how different datasets can affect the training is investigated.

To measure the prediction quality of the NNs correctly, the influence of reduction projection will be removed from measurement. In other words, the approximation error should be evaluated in the reduced space. Therefore, the reference trajectory is reduced by $\boldsymbol{Y}_{rref} = \boldsymbol{V}^{\boldsymbol{T}} \boldsymbol{Y}_{ref}$. With the prediction $\widetilde{\boldsymbol{Y}}_{\boldsymbol{r}}$ made by NN, the relative approximation error is calculated as

$$E_{\text{ann}} = \frac{1}{N_s \cdot k} \frac{|\boldsymbol{Y}_{\boldsymbol{r}\,\text{ref}} - \widetilde{\boldsymbol{Y}}_{\boldsymbol{r}}|_2}{|\boldsymbol{Y}_{\boldsymbol{r}\,\text{ref}}|_2}, \tag{24}$$

where N_s is number of snapshot simulations and k is the number of time points in each simulation.

In Tables 1–4, we list the reference tests on the left and their competitors are listed on the right. The values ΔE_{ann} stand for how much the relative prediction errors increase or decrease by using the competitive approaches. This notation is also used in the later tables. It is observed that in 45 out of 60 (3/4) test cases ROMs trained by DPS data predict better in both test cases. But unlike many other MOR

Table 1. *SPS versus DPS: heat sink model, relative error(%).*

N_r	MLP + SPS	MLP + DPS	ΔE_{ann}
(a) Constant-load test			
1	0.09	0.16	+0.07
2	0.05	0.19	+0.14
3	0.11	0.13	+0.02
4	0.10	0.16	+0.06
5	0.30	0.18	−0.12
10	0.19	0.19	−0.00
15	0.10	0.16	+0.06
20	0.11	0.15	+0.04
25	0.19	0.16	−0.04
30	0.17	0.16	−0.01
(b) Dynamic-load test			
1	0.38	0.15	−0.23
2	0.34	0.20	−0.15
3	0.33	0.15	−0.17
4	0.31	0.18	−0.13
5	0.37	0.18	−0.19
10	0.34	0.19	−0.15
15	0.29	0.17	−0.12
20	0.31	0.17	−0.14
25	0.32	0.16	−0.16
30	0.31	0.17	−0.14

Abbreviations: DPS, dynamic-parameter sampling; MLP, multilayer perceptron; SPS, static-parameter sampling.

Table 2. *SPS versus DPS: gap-radiation model, relative error(%).*

N_r	MLP + SPS	MLP + DPS	ΔE_{ann}
(a) Constant-load test			
1	0.50	0.59	+0.09
2	0.59	0.27	−0.32
3	0.25	0.28	+0.03
4	0.17	0.27	+0.10
5	0.22	0.18	−0.04
10	0.13	0.14	+0.01
15	0.17	0.22	+0.05
20	0.22	0.13	−0.10
25	0.19	0.18	−0.01
30	0.16	0.16	−0.00
(b) Dynamic-load test			
1	0.47	0.52	+0.05
2	0.63	0.19	−0.44
3	0.30	0.26	−0.04
4	0.25	0.15	−0.10
5	0.17	0.19	+0.02
10	0.27	0.19	−0.08
15	0.21	0.16	−0.04
20	0.21	0.13	−0.08
25	0.17	0.13	−0.04
30	0.19	0.12	−0.07

Abbreviations: DPS, dynamic-parameter sampling; MLP, multilayer perceptron; SPS, static-parameter sampling.

methods, the simulation error does not monotonically decrease while increasing the size of the ROM. When the ROM size is small, the simulation error is mainly caused by POD projection error. In this region, simulation error will decrease with increasing ROM size. Because the most dominant POD components are the most important physics modes of the system, by adding them the ROM knows more about the original system. However, POD components with smaller singular values might be trivial information in snapshots, for example, noise from numerical integration. These trivial information also becomes noise in training data and disturbs the learning process. This also reminds us to be careful to choose the size of ROM while using ANN-based MOR techniques.

The tests here are considered as independent tests since test datasets are provided by new NX simulation with different parameter configuration from the simulations used in the snapshots. The independent test is optimal if the two datasets originate from two different sampling strategies (Özesmi et al., 2006). Here, dynamic-load test can be considered to be optimal independent test for SPS-ROM and vice versa for DPS-ROM. As shown in Tables 1–4, the ANNs trained by DPS data have better performance on most of its optimal independent tests. But ANNs trained by SPS data also outperforms in some test cases. This result reminds us again that using dataset obtained from diverse sources should be considered.

5.2.2. Constant-time-grid learning: MLP versus RKNN

To further improve the capability of predicting ODE systems, we propose to use network architecture inspired by RK integrator in Section 3.3. Although it has been previously seen that RKNN can be used to make long-term predictions for ODE system (Wang and Lin, 1998), it is still not clear if such approach can work for POD-projected ODE systems. A possible challenge arises from the highly variable scale

Table 3. *SPS versus DPS: heat exchanger model, relative error(%).*

N_r	MLP + SPS	MLP + DPS	ΔE_{ann}
(a) Constant-load test			
1	0.20	0.27	+0.07
2	0.31	0.27	−0.04
3	0.25	0.25	−0.00
4	0.24	0.24	−0.00
5	0.13	0.16	−0.03
10	0.17	0.11	−0.05
15	0.12	0.16	+0.04
20	0.14	0.12	−0.02
25	0.15	0.15	−0.00
30	0.17	0.13	−0.04
(b) Dynamic-load test			
1	0.48	0.22	−0.26
2	0.38	0.17	−0.21
3	0.28	0.27	−0.02
4	0.31	0.21	−0.11
5	0.17	0.15	−0.02
10	0.22	0.13	−0.09
15	0.17	0.17	−0.00
20	0.18	0.14	−0.04
25	0.15	0.15	−0.00
30	0.16	0.14	−0.02

Abbreviations: DPS, dynamic-parameter sampling; MLP, multilayer perceptron; SPS, static-parameter sampling.

differences between the reduced dimensions. Typically, the reduced snapshot will have much larger component in some of the reduced dimensions (those with large singular values) than in others (small singular values). This presents a challenge for NN-based learning methods.

The structure of MLP and RKNN is the same as described in Figures 2 and 4, but in detail, the number of input neurons n_{in} depends on the size of ROM and number of system parameters, that is $n_{in} = N_r + n_\mu$. The number of neurons on each hidden layers is chosen to be 32, this value is always greater than input features, that is $N_r + n_\mu$. This is important for NN to interpret the underlying dynamics. And same as depicted in the figures, each network has one input layer, two hidden layers, and one output layer, which enables network to approximate any kind of mathematical function.

The prediction error E_{ann} is measured in the same way as in Section 5.2.1. Since it is known from the last comparison that DPS dataset is considered to be better training dataset, here both types of NNs are trained with DPS dataset. And for each test case, the time grids used for collecting snapshots and for validation have the same resolution. For these grids, 100 points are uniformly distributed on the time axis. This time resolution can be considered to be fine to the nature of the test systems. Therefore, the learning task in this section can be considered as learning from a fine time grid compared to the learning task in the next section.

In Tables 5 and 6, we can observe that RKNN can predict the solution of reduced ODE system with acceptable accuracy. But there is no indication that RKNN predicts the system behavior better in any certain type of tests. In contrast, MLP is evaluated to be better at approximating larger-size ROM. One possible explanation is that compared to a simple MLP, an RKNN can be considered as a deeper network which is harder to train (Du et al., 2019). A solution could be to train large amount of networks for each architecture but with different randomized initialization. Then evaluating the average performance of all trained networks of each architecture. Here in this paper, due to the limitation of time and computational

Table 4. *MLP versus RKNN: heat sink model, relative error(%).*

N_r	MLP + DPS	RKNN + DPS	ΔE_{ann}
(a) Constant-load test			
1	0.16	0.16	−0.01
2	0.19	0.15	−0.04
3	0.13	0.12	−0.01
4	0.16	0.16	−0.01
5	0.18	0.17	−0.01
10	0.19	0.17	−0.02
15	0.16	0.15	−0.00
20	0.15	0.17	+0.02
25	0.16	0.21	+0.05
30	0.16	0.24	+0.08
(b) Dynamic-load test			
1	0.15	0.16	+0.01
2	0.20	0.15	−0.05
3	0.15	0.12	−0.03
4	0.18	0.16	−0.02
5	0.18	0.14	−0.04
10	0.19	0.16	−0.03
15	0.17	0.16	−0.01
20	0.17	0.18	+0.02
25	0.16	0.19	+0.04
30	0.17	0.25	+0.08

Abbreviations: DPS, dynamic-parameter sampling; MLP, multilayer perceptron; RKNN, Runge–Kutta neural network.

resource, we only train 10 networks for each type of architecture, which might not be sufficient to eliminate the random effect.

5.2.3. Coarse-time-grid learning: EENN versus RKNN

Although the results of the previous section are inconclusive to demonstrate the advantages of RKNN over MLP, here we will analyze a practical advantage of learning the system dynamics (learning $\boldsymbol{f}(\cdot)$) compared to directly learning the evolution of the system state. In the former case, the time step size for prediction can be chosen independently of the time step used to take the snapshots. In this section, we use a training data set in which snapshots are sampled coarsely, that is with a large time interval, to build new ROMs. Then we evaluate the capability of these ROMs to predict the time evolution of the system under new parameters and using finer integration time steps. Such a scenario is relevant in the context of MOR when the FOMs are very large, as time and memory constraints might limit the amount of (full order) input data available to provide to the MOR methods.

MLPs (as defined in Section 3.1) cannot be applied to this test as we cannot use an independent time step for evaluation. Therefore, we consider only EENNs and RKNNs. For examples in which the training and evaluation time steps are equivalent, we expect EENNs to be equivalent to MLPs (Figure 11).

In the experiments, we prepare two dataset. For the dataset A, $K = 25$ snapshots are collected in each snapshot simulation with DPS. For the purpose of validation, ROMs are asked to predict in the same time span but with $L = 25, 50, 100, 200, 300, 500, 1{,}000$ steps. To be more specific, if the step size used in training dataset is τ_s, the step size used in validation is $\tau_d = \tau_s, \tau_s/2, \ldots, \tau_s/20, \tau_s/40$. In the dataset B, $K = 100$ snapshots are collected in each snapshot simulation. The trained ROMs will predict in the same time span with $L = 100, 200, 300, 500, 1{,}000$ steps. The reduced data in both training sets has the size of 10. The

Table 5. *MLP versus RKNN: gap-radiation model, relative error(%).*

N_r	MLP + DPS	RKNN + DPS	ΔE_{ann}
(a) Constant-load test			
1	0.59	0.47	−0.13
2	0.27	0.24	−0.03
3	0.28	0.21	−0.07
4	0.27	0.19	−0.08
5	0.18	0.21	+0.02
10	0.14	0.19	+0.04
15	0.22	0.19	−0.03
20	0.13	0.19	+0.06
25	0.18	0.24	+0.06
30	0.16	0.21	+0.05
(b) Dynamic-load test			
1	0.52	0.38	−0.13
2	0.19	0.25	+0.06
3	0.26	0.26	+0.01
4	0.15	0.25	+0.10
5	0.19	0.16	−0.03
10	0.19	0.21	+0.02
15	0.16	0.16	−0.01
20	0.13	0.19	+0.06
25	0.13	0.19	+0.06
30	0.12	0.20	+0.08

Abbreviations: DPS, dynamic-parameter sampling; MLP, multilayer perceptron; RKNN, Runge–Kutta neural network.

ROMs with such a size are observed to be stable with the POD projection error according to the previous tests. And we use DPS to collect the data since it generally collects more information.

Figure 12 presents the results of this comparison. Each of the plots corresponds to a different test model from Section 4 and they show the relative errors, calculated according to Equation (24) for prediction at different time step sizes. For the reference (training) time step, the errors are small and comparable for both EENN and RKNN, which is consistent with the observations in Section 5.2.2. For smaller evaluation time steps, the errors become larger. We speculate that the learned R.H.S. still contains information from the time step size used during training. However, we can clearly see that this effect is consistently smaller for RKNN-MOR. This suggests that the use of the higher order numerical scheme during training can provide a better quality inferred R.H.S.

To further explore this hypothesis we can perform an additional test of the trained ROMs. As both EENN and RKNN learn the right side $\boldsymbol{f}(\cdot)$ of Equation (8), we can integrate the ROM with a different numerical scheme as the training. Results of these tests are presented in Figure 13, where we plot these tests alongside the data in Figure 12. The new data includes the following: (a) the error incurred from integrating the EENN-ROM using RK scheme and (b) the error incurred from integrating the RKNN-ROM using EE scheme. The same qualitative behavior is observed for all test models. Interestingly, the error is almost always largest and roughly independent of the step size when using RK scheme to integrate the EENN-ROM. For RKNN-ROMs evaluated using EE scheme, the trend is quite different. For the original (large) time step sizes, the errors are always larger than the reference RKNN results. However, for smaller evaluation time steps the error approaches the same values as the result of integrating using RK scheme.

We can interpret these trends as follows: for the RKNN-ROM (red curves), we see that evaluating using EE scheme (circle markers) gives larger errors than RK scheme (cross markers) for large time steps, but

Table 6. *MLP versus RKNN: heat exchanger model, relative error(%).*

N_r	MLP + DPS	RKNN + DPS	ΔE_{ann}
(a) Constant-load test			
1	0.27	0.27	+0.01
2	0.27	0.13	−0.13
3	0.25	0.15	−0.10
4	0.24	0.20	−0.04
5	0.16	0.13	−0.03
10	0.11	0.18	+0.07
15	0.16	0.20	+0.05
20	0.12	0.16	+0.04
25	0.15	0.17	+0.02
30	0.13	0.17	+0.04
(b) Dynamic-load test			
1	0.22	0.18	−0.04
2	0.17	0.13	−0.04
3	0.27	0.15	−0.11
4	0.21	0.19	−0.02
5	0.15	0.15	+0.00
10	0.13	0.19	+0.06
15	0.17	0.21	+0.04
20	0.14	0.17	+0.03
25	0.15	0.18	+0.03
30	0.14	0.18	+0.04

Abbreviations: DPS, dynamic-parameter sampling; MLP, multilayer perceptron; RKNN, Runge–Kutta neural network.

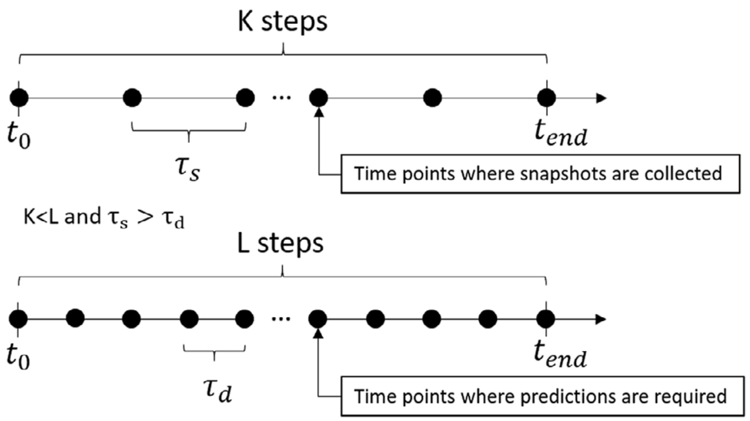

K steps
t_0
τ_s
t_{end}
Time points where snapshots are collected
K<L and $\tau_s > \tau_d$
L steps
t_0
τ_d
t_{end}
Time points where predictions are required

Figure 11. *Time grid for collecting snapshots and time grid for prediction with reduced order model (ROM).*

that the error converges for smaller time steps. This is consistent with having an accurate R.H.S. and integrating it using a higher or lower order integration scheme. For the EENN-ROM (blue curves), the error is almost always larger than for RKNN-ROM and its convergence trend also contradicts the result that we would observe with a real analytical R.H.S. Additionally, the errors become small only while matching the evaluation conditions with the training conditions. This is consistent with our hypothesis

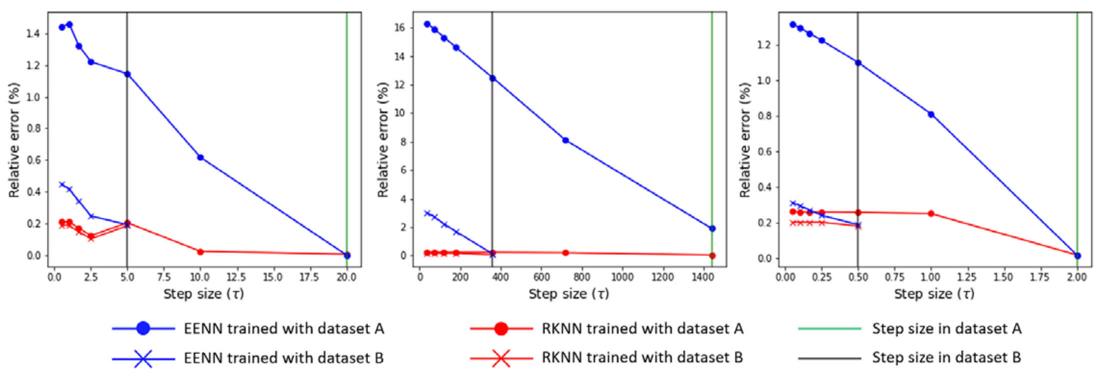


Figure 12. *The graph from the left to the right belongs to the heat sink model, the gap-radiation model and the heat exchanger model respectively. The reduced order model (ROMs) are generated from the coarse and the fine time grids respectively. The prediction is made with time steps which are always finer than the step sizes used in training data. The original step sizes used in two training snapshots are marked with green (dataset A) and black (dataset B) vertical lines.*

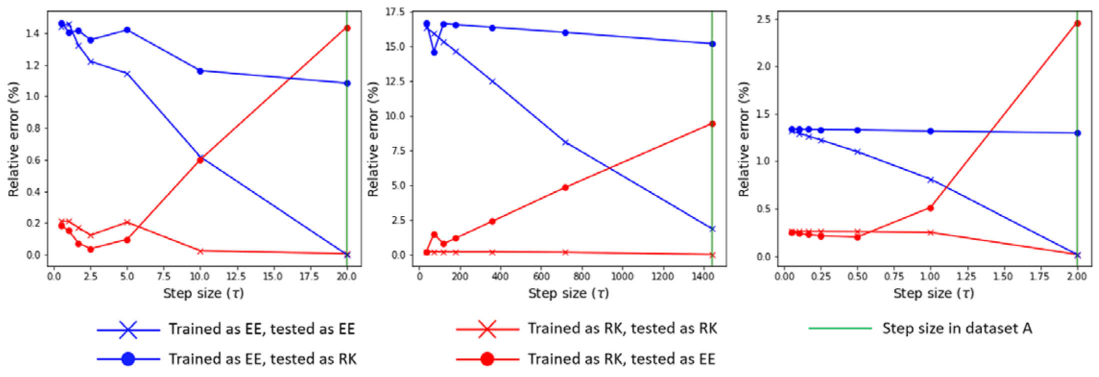


Figure 13. *The graph from the left to the right belongs to the heat sink model, the gap-radiation model and the heat exchanger model respectively. The tests aim to evaluate the accuracy of the learned reduced order model (ROMs) in.*

that the R.H.S. learned by EENNs are more influenced by the time step size in the training data, therefore having less flexibility for changing it.

In summary, we believe that these tests strongly suggest that the use of higher order integration schemes during training can noticeably improve the quality of the dynamics learned in the reduced space, especially when the input data is sampled sparsely.

6. Conclusions

In this paper, we investigated different varieties of MOR based on ANN. In the first part, we looked at different ways of generating the training data, namely SPS versus DPS. In the second part, we study the influence of using different NN architectures. We compared MLP NNs that directly learn the evolution of the system state with RKNN which learn the R.H.S. of the systems. Besides, we compared RKNN versus EENN which learns the system R.H.S. similarly but with lower-order numerical scheme, with a focus on the effect on learning reduced models from sparsely sampled data.

As discussed in Section 5.2.1, DPS shows positive influence on the quality of ROM. Based on observation, adding DPS snapshots into the dataset can enrich the dynamics contained in snapshots. This can help with constructing better POD basis. Moreover, NN trained by DPS dataset is more sensitive to system's parameters. However, it is also seen that in some specific tests, network trained by DPS data performs worse than networks trained by SPS data, which are usually found to be in the constant-load tests. However, the direct comparison of RKNN and MLP did not show clear advantage of either architecture under the conditions of the tests in Section 5.2.2.

The advantage of RKNN could actually be demonstrated by analyzing a test case in which the training dataset is sampled coarsely in time. Here, we observed, that RKNN can predict with different step sizes without significant increment of the prediction error. In contrast, this was not the case for EENN. Our results also suggest that embedding a higher order integration scheme can help learning the systems dynamics more reliably and independently of the time step in the input data. This result is relevant for engineering applications, where memory and computational time constraints can limit the amount of data made available for MOR.

Although this MOR framework was tested only on thermal and thermal-flow models, it could be easily adapted to other kinds of models, since it is nonintrusive and only relies on measured data to learn the dynamics of the system. However, there is still more work to be done before these methods can be applied to big scale and more general models.

Some further potential improvements for the approach discussed in this work include:

- Investigating the influence of using SPS–DPS-mixed snapshots.
- Investigating different methods of constructing reduced space other than POD, for example, auto-encoder.
- Implementing implicit integration scheme with NNs as in Anderson et al. (1996) allowing more accurate long-term prediction.
- Implementing integration scheme with multiple history states as part of input, which might also enable more stable long-term prediction.

Acknowledgments. The authors are grateful to Siemens for sponsoring this project. The authors also thank Prof. Benjamin Peherstorfer and Lukas Failer for their valuable suggestions for the research.

Funding Statement. This work is part of the research undertaken by the authors in COMPAS which is an ITEA-EUREKA project co-funded by the Federal Ministry of Education and Research of Germany (BMBF).

Competing Interests. Q.Z., J.L., and D.H. are employed at company Siemens AG.

Data Availability Statement. To get access to research data, please contact one of the authors with your request.

Author Contributions. Conceptualization: D.H. and H.-J.B.; Methodology: D.H., Q.Z., and J.L.; Data curation: Q.Z.; Data visualization: Q.Z.; Writing-original draft: Q.Z., J.L., and. D.H. All authors approved the final submitted draft.

References

Agudelo OM, **Espinosa JJ and De Moor B** (2009) Acceleration of nonlinear POD models: a neural network approach. In *2009 European Control Conference (ECC)*. Budapest, Hungary: IEEE, pp. 1547–1552.

Anderson J, **Kevrekidis I and Rico-Martinez R** (1996) A comparison of recurrent training algorithms for time series analysis and system identification. *Computers & Chemical Engineering 20*, S751–S756.

Antoulas A (2005) *Approximation of Large-Scale Dynamical Systems*. Philadelphia, USA: Society for Industrial and Applied Mathematics.

Bui-Thanh T (2007). Model-constrained optimization methods for reduction of parameterized large-scale systems. PhD Thesis, Massachusetts Institute of Technology.

Chaturantabut S and Sorensen DC (2010). Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing 32*(5), 2737–2764.

Coifman RR, **Lafon S**, **Lee AB**, **Maggioni M**, **Nadler B**, **Warner F and Zucker SW** (2005) Geometric diffusions as a tool for harmonic analysis and structure definition of data: diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America 102*(21), 7426–7431.

Du Q and Fowler JE (2007) Hyperspectral image compression using JPEG2000 and principal component analysis. *IEEE Geoscience and Remote Sensing Letters 4*(2), 201–205.

Du S, **Lee J**, **Li H**, **Wang L and Zhai X** (2019) Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*. Long Beach, USA: PMLR, pp. 1675–1685.

Glorot X, **Bordes A and Bengio Y** (2011) Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ft. Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, pp. 315–323.

Haasdonk B, **Dihlmann M and Ohlberger M** (2011) A training set and multiple bases generation approach for parameterized model reduction based on adaptive grids in parameter space. *Mathematical and Computer Modelling of Dynamical Systems 17* (4), 423–442.

Hartmann D, **Herz M and Wever U** (2018) Model order reduction a key technology for digital twins. In *Reduced-Order Modeling (ROM) for Simulation and Optimization*. Springer, pp. 167–179.

He K, **Zhang X**, **Ren S and Sun J** (2016) Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA: IEEE, pp. 770–778.

Helton JC and Davis FJ (2003) Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. *Reliability Engineering & System Safety 81*(1), 23–69.

Hinze M and Volkwein S (2005) Proper orthogonal decomposition surrogate models for nonlinear dynamical systems: Error estimates and suboptimal control. In *Dimension Reduction of Large-Scale Systems*. Berlin, Heidelberg: Springer, pp. 261–306.

Josse J and Husson F (2012) Selecting the number of components in principal component analysis using cross-validation approximations. *Computational Statistics & Data Analysis 56*(6), 1869–1879.

Kani JN and Elsheikh AH (2017) DR-RNN: a deep residual recurrent neural network for model reduction. *arXiv preprint arXiv: 1709.00939.*

Kassab A, **Divo E**, **Heidmann J**, **Steinthorsson E and Rodriguez F** (2003) BEM/FVM conjugate heat transfer analysis of a three-dimensional film cooled turbine blade. *International Journal of Numerical Methods for Heat & Fluid Flow 13*, 581–610.

Kingma DP and Ba J (2014) Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Kosmatopoulos EB, **Polycarpou MM**, **Christodoulou MA and Ioannou PA** (1995) Highorder neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks 6*(2), 422–431.

Lappano E, **Naets F**, **Desmet W**, **Mundo D and Nijman E** (2016) A greedy sampling approach for the projection basis construction in parametric model order reduction for structural dynamics models. In *Proceedings of ISMA 2016—International Conference on Noise and Vibration Engineering and USD2016—International Conference on Uncertainty in Structural Dynamics*, Leuven, Belgium: KU Leuven pp. 3563–3571.

Mohan AT and Gaitonde DV (2018) A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks. *arXiv preprint arXiv:1804.09269.*

Özesmi SL, **Tan CO and Özesmi U** (2006) Methodological issues in building, training, and testing artificial neural networks in ecological applications. *Ecological Modelling 195*(1–2), 83–93.

Pan S and Duraisamy K (2018) Long-time predictive modeling of nonlinear dynamical systems using neural networks. *Complexity 2018*, 4801012.

Paszke A, **Gross S**, **Chintala S**, **Chanan G**, **Yang E**, **DeVito Z**, **Lin Z**, **Desmaison A**, **Antiga L and Lerer A** (2017) Automatic differentiation in PyTorch.

Pawar S, **Rahman S**, **Vaddireddy H**, **San O**, **Rasheed A and Vedula P** (2019) A deep learning enabler for nonintrusive reduced order modeling of fluid flows. *Physics of Fluids 31*(8), 085101.

Peherstorfer B and Willcox K (2016) Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering 306*, 196–215.

Prechelt L (1998). Early stopping-but when?. In *Neural Networks: Tricks of the Trade*. Berlin, Heidelberg: Springer, pp. 55–69.

Qin T, **Chen Z**, **Jakeman JD and Xiu D** (2021) Data-driven learning of nonautonomous systems. *SIAM Journal on Scientific Computing 43*(3), A1607–A1624.

Qin T, **Wu K and Xiu D** (2019). Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics 395*, 620–635.

Rasheed A, **San O and Kvamsdal T** (2019) Digital twin: Values, challenges and enablers. *arXiv preprint arXiv:1910.01719.*

Rico-Martinez R and Kevrekidis IG (1993). Continuous time modeling of nonlinear systems: A neural network-based approach. In *IEEE International Conference on Neural Networks, 1993*. San Francisco, CA, USA: IEEE, pp. 1522–1525.

Rumelhart DE, **Hinton GE and Williams RJ** (1986) Learning representations by back-propagating errors. N*ature 323*(6088), 533.

Ubbiali S (2017). Reduced order modeling of nonlinear problems using neural networks.

Valle S, **Li W and Qin SJ** (1999). Selection of the number of principal components: the variance of the reconstruction error criterion with a comparison to other methods. *Industrial & Engineering Chemistry Research 38*(11), 4389–4401.

van der Maaten L and Hinton G (2008) Visualizing data using t-SNE. *Journal of Machine Learning Research 9*, 2579–2605.

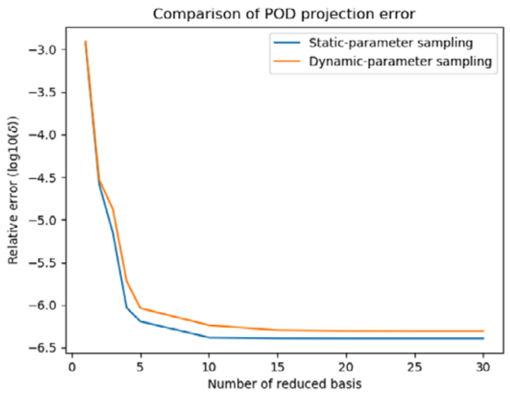
Wang Q, **Ripamonti N and Hesthaven JS** (2020). Recurrent neural network closure of parametric POD-Galerkin reduced-order models based on the Mori-Zwanzig formalism. *Journal of Computational Physics 410*, 109402.

Wang Y-J and Lin C-T (1998). Runge-Kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks 9*(2), 294–307.

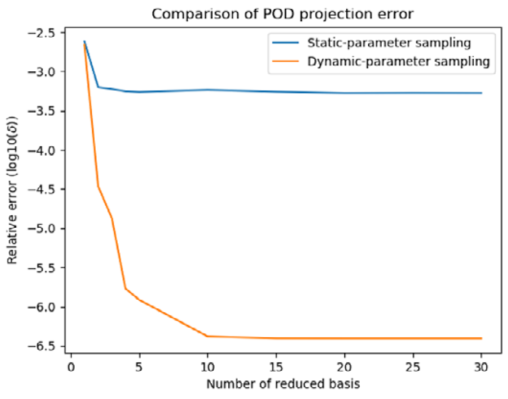
Willcox K and Peraire J (2002). Balanced model reduction via the proper orthogonal decomposition. *AIAA Journal 40*(11), 2323–2330.

Wong T-T, **Luk W-S and Heng P-A** (1997) Sampling with Hammersley and Halton points. *Journal of Graphics Tools 2*(2), 9–24.

Appendix: Additional Figures

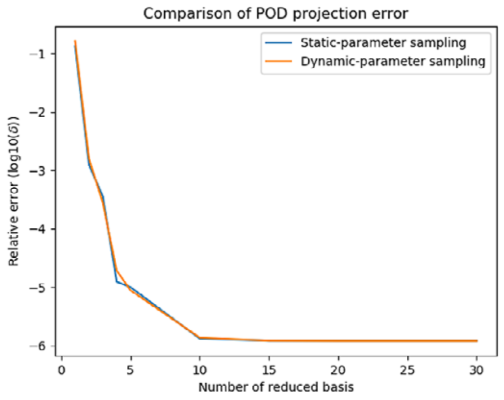


(a) Constant load test

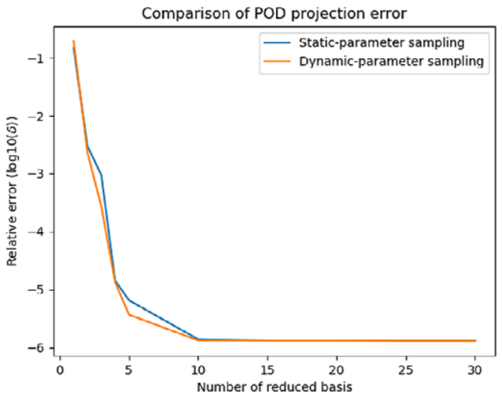


(b) Dynamic load test

Figure A1. *Projection error: heat sink model.*

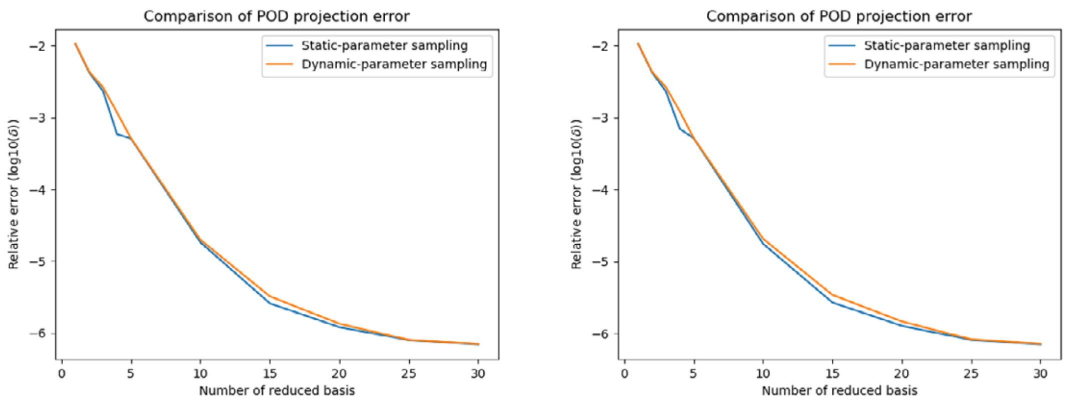


(a) Constant load test



(b) Dynamic load test

Figure A2. *Projection error: gap-radiation model.*



(a) Constant load test (b) Dynamic load test

Figure A3. *Projection error: heat exchanger model.*