A Regularized Real-Time Integrator for Data-Driven Control of Heating Channels

Chady Ghnatios 1,*, Victor Champaney 2, Angelo Pasquale 2,3 and Francisco Chinesta 4,5

1 Department of Mechanical Engineering, Notre Dame University-Louaize, Zouk Mosbeh, P.O. Box 72, Lebanon
2 ESI Group Chair at PIMM Laboratory, Arts et Métiers Institute of Technology, 151 Boulevard de l’Hôpital, 75013 Paris, France
3 ESI Group Chair at LAMPA Laboratory, Arts et Métiers Institute of Technology, 2 Boulevard du Ronceray, BP 93525, CEDEX 01, 49035 Angers, France
4 PIMM Laboratory, Arts et Métiers Institute of Technology, CNRS, Cnam, HESAM Université, 151 Boulevard de l’Hôpital, 75013 Paris, France
5 ESI Group, 3 bis Rue Saarien, 94528 Rungis, France
* Correspondence: cghnatios@ndu.edu.lb

Abstract: In many contexts of scientific computing and engineering science, phenomena are monitored over time and data are collected as time-series. Plenty of algorithms have been proposed in the field of time-series data mining, many of them based on deep learning techniques. High-fidelity simulations of complex scenarios are truly computationally expensive and a real-time monitoring and control could be efficiently achieved by the use of artificial intelligence. In this work we build accurate data-driven models of a two-phase transient flow in a heated channel, as usually encountered in heat exchangers. The proposed methods combine several artificial neural networks architectures, involving standard and transposed deep convolutions. In particular, a very accurate real-time integrator of the system has been developed.

Keywords: machine learning; artificial intelligence; integrator; data-driven modeling

1. Introduction

Most phenomena in engineering are described by sequential data. In the context of data-driven simulation and control [1], time series classification and forecasting play therefore a fundamental role. A huge variety of applications, such as financial mathematics, language and speech recognition, weather forecasting, physical and chemical unsteady systems, are extensively discussed in the literature [2–5].

In the last decade, data-driven approximations of the Koopman operator have successfully been applied to build reduced-models of complex dynamical systems, allowing their identification and control [6–9]. Several approaches are based on combining algorithms derived from the dynamic mode decomposition–DMD–with deep learning–DL–techniques, as discussed in [10,11] and references therein.

Many algorithms have been applied in artificial intelligence–AI–for the estimation of output time series from input time series. Among them, convolution and recurrent neural networks–CNNs, RNNs– [12–14], time delay neural networks–TDNNs– [15–17], nonlinear autoregressive neural networks–NARNETs– [18,19] and nonlinear autoregressive neural networks with exogenous inputs–NARXNETs– [20–24].

In this work, several surrogates based on standard and transposed convolution neural networks–CNNs– [25–27] are applied to a thermohydraulic problem in the context of heat exchangers involving change of phase [28–30]. The problem will be described in detail later. It basically consists in fixing the input heat power $u(t)$ injected into a two-phase flow and monitoring some quantities $y(t)$ at the outlet of a pipeline in which the gas-liquid mixture is confined and flows.
Mathematically speaking, let us consider an input and output time series, respectively $u$ and $y$, tracking $N_t$ time instants. The problem is the estimation of the non-linear transfer function $h$ such that $y = h(u)$. Learning the function $h$ allows the real-time integration of a newly defined input to know the impact on some outputs of interest.

A first approach is based on exploiting a parametrization of the input function $u(t)$, through a few values $\{U_1, U_2, \ldots, U_K\}$. In this case, the modeling framework is non-incremental, since the whole output time series is computed from the inputs $\{U_1, U_2, \ldots, U_K\}$, that is:

$$y = h(U_1, U_2, \ldots, U_K).$$  \hfill (1)

The estimation of $h$ will be done through the use of artificial neural networks. However, since the output consists of $N_t$ evaluations, while the input of $K < N_t$ parameters, a transposed convolution (also known as fractionally-strided convolution) is employed to reach the right dimension at the output layer. As a matter of fact, transposed convolutions increase (up sample) the spatial dimensions of intermediate feature maps, reversing down sample operations obtained by classical convolutions [25].

A second approach consists of considering an incremental modelling framework, in which the model (also known as integrator) computes the next value of the output sequence from a given number of previous inputs and outputs. Such approach has some similarities with the NARXNETs [18]. Denoting with $y_i = y(t_i)$, for $i = 1, \ldots, N_t$, we can write:

$$y_i = h(y_{i-1}, y_{i-2}, \ldots, y_{i-k}, u_i, u_{i-1}, \ldots, u_{i-k+1}),$$  \hfill (2)

where the parameter $k \geq 0$ is a delay term, known as the process dead time. We can also note $U = y_{\text{past}} \cup u$, $y_{\text{past}}$ being the known $y$ outputs used to predict future values of $y$. A neural-network architecture based on a convolution approach can then be defined. Two non-incremental models (for different values of the delay coefficient $k$) will be compared, in terms of speed and accuracy.

The paper aims to build a data-driven model that is capable of reproducing the data with high fidelity, while possibly performing the integrator task over a large time frame. First of all, the model of the addressed problem and the simulation data structure are described in Section 2. Section 3 presents three different surrogates:

- A non-incremental modelling framework is presented in Section 3.1, which predicts the whole time frame in a single shot
- Two incremental alternatives are presented in Section 3.2. These alternatives used different memory size to predict the future variation in the quantities of interest.
- The incremental surrogate model with large memory is then used as an integrator to predict the variation over a large time series

Finally, Section 4 addresses the conclusion summarizing the results and highlighting the main differences among the learned models.

2. Problem Statement

High-fidelity numerical simulations of a thermohydraulic problem are performed using the software CATHARE for nuclear safety analysis [31–33]. The simulation consists into a two-phase (steam-water) flow moving upward against gravity into a channel heated by hot walls, as sketched in Figure 1. The model input is the time-varying thermal power $q(t)$ injected into the fluid through the walls, while the outputs are the time histories of several fluid properties at the channel outlet. The simulation covers the interval $[0, 100]$ s, with a time step $\Delta t = 0.1$ s.
Figure 1. Fluid channel with wall surface heating, controlled by the heat power density law \( q(t) \). The green point represents the location of the measurement node at the outlet.

Using the bold notation for vectors (sequences of 1000 elements corresponding to the time discretization), the quantities of interest are the steam fraction in the flowing system \( A \), the liquid velocity \( V \), the steam temperature \( T_G \) and the liquid temperature \( T_L \). This work aims to produce a surrogate model \( h \) for inferring the outputs as a function of the input variable \( q \):

\[
\begin{pmatrix}
    A \\
    V \\
    T_G \\
    T_L \\
\end{pmatrix}
= h(q)
\quad (3)
\]

The surrogate model is built in a non-intrusive manner, defining a Design of Experiments–DoE–on the power density and combining the outputs of the corresponding offline simulations.

A user-defined power control law allows to assign the value of the provided power density at specific time instants. Therefore, the input function \( q(t) \) is defined through a few values \( Q_i = q(t_i) \), normalized by the maximum power density \( Q_{\text{max}} = 8.4 \times 10^8 \text{ W/m}^3 \), and then linearly interpolating in-between. In particular, the time instants (in seconds) \( \{0, 1, 25, 50, 75, 100\} \) are considered for the power changes. The corresponding values \( Q_i \), for \( i = 0, \ldots, 5 \), define a parametrization of the input function \( q(t) \).

A DoE of 23 data points is established using different combinations of the values \( Q_i \), as listed in Table 1. For the rest of this work, the last 4 datasets are considered in the testing set and therefore not fed into the training loop of the built surrogate model. Note that some of the values are fixed (\( Q_0 = 0, Q_1 = Q_5 = 1 \)) thus the model input parameters are only \( Q_2, Q_3 \) and \( Q_4 \). Figure 2 shows an example of the heat power density \( q(t) \) defined from values \( Q_i \).

Table 1. Available plan of experiment (DoE) and sequences of control points values, as a fraction of \( Q_{\text{max}} = 8.4 \times 10^8 \text{ W/m}^3 \).

| Sequence Number | \( Q_0 \) | \( Q_1 \) | \( Q_2 \) | \( Q_3 \) | \( Q_4 \) | \( Q_5 \) |
|-----------------|----------|----------|----------|----------|----------|----------|
| 1               | 0        | 1        | 0        | 0        | 0        | 1        |
| 2               | 0        | 1        | 1        | 0        | 0        | 1        |
| 3               | 0        | 1        | 0        | 1        | 0        | 1        |
| 4               | 0        | 1        | 0        | 0        | 1        | 1        |
Table 1. Cont.

|   | 0   | 1   | 0   | 1   | 1   | 1   |
|---|-----|-----|-----|-----|-----|-----|
| 5 | 0   | 1   | 0   | 1   | 1   | 1   |
| 6 | 0   | 1   | 1   | 0   | 1   | 1   |
| 7 | 0   | 1   | 1   | 1   | 0   | 1   |
| 8 | 0   | 1   | 1   | 1   | 1   | 1   |
| 9 | 0   | 1   | 0.19 | 0.07 | 0.9 | 1   |
|10 | 0   | 1   | 0.06 | 0.8  | 0.28 | 1   |
|11 | 0   | 1   | 0.91 | 0.76 | 0.58 | 1   |
|12 | 0   | 1   | 0.69 | 0.19 | 1   | 1   |
|13 | 0   | 1   | 0.12 | 0.57 | 0.38 | 1   |
|14 | 0   | 1   | 0.2  | 0.67 | 0.02 | 1   |
|15 | 0   | 1   | 0.99 | 0.92 | 0.47 | 1   |
|16 | 0   | 1   | 0.55 | 0.94 | 0.74 | 1   |
|17 | 0   | 1   | 0.47 | 0.24 | 0.83 | 1   |
|18 | 0   | 1   | 0.76 | 0.36 | 0.18 | 1   |
|19 | 0   | 1   | 0.82 | 0.49 | 0.4  | 1   |
|20 | 0   | 1   | 0.66 | 0.66 | 0.72 | 1   |
|21 | 0   | 1   | 0.31 | 0.43 | 0.6  | 1   |
|22 | 0   | 1   | 0.37 | 0.31 | 0.11 | 1   |
|23 | 0   | 1   | 0.44 | 0.05 | 0.2  | 1   |

Figure 2. Example of provided heat power density, as a fraction of $Q_{max} = 8.4 \times 10^8$ W/m$^3$ (this corresponds to the run 9 of the DoE).

A sample set of results, of the available datasets number 1, 2, 3 and 4 of Table 1, is shown in Figure 3 for illustration purposes.
3. Surrogate Modeling of the Data

Several models are tested and in what follows we explain the results of three of them. Before any training, the variables are normalized by subtracting their corresponding means and dividing by the variation amplitude $\text{max} - \text{min}$.

3.1. A Non-Incremental Approach

In this section, we present a non-incremental modelling framework, aiming to reproduce all the measured 1000 steps time sequence while considering only the three variables $Q_2$, $Q_3$ and $Q_4$. Therefore, the first model $\mathcal{H}_1$ aims to create an output of the form:

$$
\begin{pmatrix}
\hat{A} \\
\hat{V} \\
\hat{T}_G \\
\hat{T}_L
\end{pmatrix} = \mathcal{H}_1(Q_2, Q_3, Q_4),
$$

with $\hat{}$ being the approximation of the data, output of the surrogate model. In this section, since we are using a non-incremental approach, each dataset is nothing but a single data-point, for which we evaluate a single output. We use a neural network based on a transposed convolution approach [25–27]. The network starts with an input of 3 values, for instance $(Q_2, Q_3, Q_4)$, and propagate the variation of the input variables into the required output. The architecture of the used network is illustrated in Table 2 and schematically represented through the diagram in Figure 4. The first layer in the designed neural network transforms the input into a dimension suitable for beginning a convolution transpose. The second layer reshapes the inputs into a simulated 2D picture of dimension $8 \times 4$, with 240 filters. The subsequent layers reduce the number of filtering while increasing the size of the first dimension of the tensor, through multiplying by 5 on each layer, reaching therefore a dimension of $1000 \times 4$ on layer 6, the exact dimension of the required 4 variables output.
Table 2. Structure of the deep convolution transpose neural network used for the fitting of $H_1$. 
\( \text{tanh} \) stands for the hyperbolic tangent activation function, \( \text{selu} \) stands for the scaled exponential linear unit \([34]\), while \( \text{linear} \) stands for linear activation or no activation function. CT stands for “convolution transpose”.

| Layers | Shape | Activation |
|--------|-------|------------|
| 1      | Fully connected dense layer with 7680 neurons | \( \text{tanh} \) |
| 2      | Reshape layer into a 3D Tensor of shape \((8 \times 4 \times 240)\) | No activation |
| 3      | 2D CT, 120 filters, kernel \((5 \times 1)\), strides \((5 \times 1)\), “same” padding | \( \text{selu} \) |
| 4      | 2D CT, 60 filters, kernel \((5 \times 1)\), strides \((5 \times 1)\), “same” padding | \( \text{selu} \) |
| 5      | 2D CT, 30 filters, kernel \((5 \times 1)\), strides \((5 \times 1)\), “same” padding | \( \text{selu} \) |
| 6      | 2D CT, 1 filter, kernel \((1 \times 1)\), strides \((1 \times 1)\), “same” padding | \( \text{linear} \) |

Figure 4. Graphical diagram of the deep convolution neural network used for the fitting of $H_1$. All convolutions employ “same” padding (also called “half” padding). If the input and output of a transpose convolution have shapes \( I_1 \times I_2 \times I_3 \) and \( O_1 \times O_2 \times O_3 \), respectively, then the dimensions are obtained as \( O_i = s_i I_i \), where \( s_i \) denotes the stride along the dimension \( i \), for \( i = 1, 2 \). The dimension \( O_3 \) is determined by the number of filters.

The neural network is trained over 5000 epochs, with a batch size of 10 datasets, knowing that we have only 19 available datasets. The selected loss function is the mean square error, without regularization, and the optimization algorithm is Adam \([35]\) with an initial learning rate \( \alpha = 10^{-4} \). The results for a selected training dataset are shown in Figure 5, whereas the results for the selected testing datasets are illustrated for datasets 20 and 22 in Figures 6 and 7, respectively.
Figure 5. Sample results of $H_1$ surrogate model for the training datasets number 10, as shown in Table 1.

Figure 6. Sample results of $H_1$ surrogate model for the testing datasets number 20, as shown in Table 1.
Figure 7. Sample results of $H_1$ surrogate model for the testing datasets number 22, as shown in Table 1. Note the representation scale in panel (c).

The relative errors generated by the non-incremental approach are illustrated in Table 3. Since the output $A$ contains several zero values, the relative error $E_A$ is normalized with respect to the maximum value of $A$:

$$E_A = \frac{\|A - \hat{A}\|_2}{\|A\|_\infty},$$

with $A$ being the measured data and $\hat{A}$ the surrogate model output. The normalization is standard for any other variable $v$:

$$E_v = \frac{\|v - \hat{v}\|}{\|v\|_2},$$

where, similarly, $v$ denotes the measured data and $\hat{v}$ the surrogate model output and the symbol $\odot$ the Hadamard division operator. The results are excellent in either training or testing sets.

Table 3. Mean relative errors of the trained non-incremental model $H_1$.

| Model $H_1$ | $E_A$ | $E_V$ | $E_{T_G}$ | $E_{T_L}$ |
|-------------|-------|-------|-----------|-----------|
| Training sets | 0.058% | 0.735% | $1.7 \times 10^{-5}$% | 0.037% |
| Testing sets | 0.14% | 1.61% | $9.0 \times 10^{-5}$% | 0.265% |

3.2. Incremental Approaches

Using an incremental approach consists in learning the integrator that allows to update the state of the system at each time step. The model is built by first decomposing the datasets into a sequence chain of length $t_s$. Increasing the chain size $t_s$, by considering therefore a higher number of time steps in the variation history, will definitely improve
the results, by jeopardizing the computation time required for the prediction of the data point. Two incremental models are tested, the first one considers $t_s = 4$, and is an excellent predictor of the next output of the considered sequences but fails as dynamical integrator as will be discussed later, whereas the second model uses $t_s = 20$, with a regularization, for improving stability, that performs correctly as integrator.

### 3.2.1. A Fast Predictor

In this section we explain the possibility of using a fast algorithm based on convolution layers, to predict the next outcome of the sequence at hand. The model will take as an input the variables $(A, V, T_C, T_L)$ at time steps $i-1, i-2, i-3$ and $i-4$, as well as the input $q$ at time steps $i, i-1, i-2$ and $i-3$, to predict the quantities of interest at time step $i$. For instance, we can write the second trained surrogate model $H_2$ as follows:

$$
\begin{pmatrix}
\tilde{A}_i \\
\tilde{V}_i \\
T_{Gi} \\
T_{Li}
\end{pmatrix} = H_2
\begin{pmatrix}
A_{i-1} \\
A_{i-2} \\
A_{i-3} \\
A_{i-4}
\end{pmatrix},
\begin{pmatrix}
V_{i-1} \\
V_{i-2} \\
V_{i-3} \\
V_{i-4}
\end{pmatrix},
\begin{pmatrix}
T_{Gi-1} \\
T_{Gi-2} \\
T_{Gi-3} \\
T_{Gi-4}
\end{pmatrix},
\begin{pmatrix}
T_{Li-1} \\
T_{Li-2} \\
T_{Li-3} \\
T_{Li-4}
\end{pmatrix},
\begin{pmatrix}
q_i \\
q_{i-1} \\
q_{i-2} \\
q_{i-3}
\end{pmatrix}.
\tag{7}
$$

The considered network to approximate the surrogate model $H_2$ is summarized in Table 4 and schematized in Figure 8. The input of the network is a 3D tensor of dimension $(4 \times 5 \times 1)$, for 4 time steps and 5 variables. The selected optimization algorithm is Adam with the loss function being the mean square errors. The available data after reformulation into a sequence of 4 steps has the size of $19 \times 996$ inputs for the training sets and $4 \times 996$ for the testing set. A batch size of 996 is selected with 1500 epochs for the training. The initial learning rate was set to $a = 10^{-4}$.

Table 4. Structure of the deep convolution neural network used for the fitting of $H_2$. \textit{tanh} stands for the hyperbolic tangent activation function, \textit{selu} stands for the scaled exponential linear unit [34], while \textit{linear} stands for linear activation or no activation function.

| Layers | Shape | Activation |
|--------|-------|------------|
| 1      | 2D convolution, 60 filters, kernel $(2 \times 1)$, strides $(2 \times 1)$, "same" padding | selu |
| 2      | 2D convolution, 120 filters, kernel $(2 \times 1)$, strides $(1 \times 1)$, "same" padding | selu |
| 3      | A flatten layer, reshapes all inputs into a single vector | no activation |
| 4      | Fully connected dense layer with 250 neurons | selu |
| 5      | Fully connected dense layer with 125 neurons | selu |
| 6      | Fully connected dense layer with 70 neurons | selu |
| 7      | Fully connected dense layer with 25 neurons | tanh |
| 8      | Fully connected dense layer with 4 neurons | linear |

The results for a selected training dataset are shown in Figure 9, whereas the results for the selected testing datasets are illustrated for datasets 20 and 22 in Figures 10 and 11, respectively. The relative errors are shown in Table 5, computed using Equations (5) and (6), as described in Section 3.1.

Model $H_2$ shows an excellent ability to reproduce the datasets when the input is always taken from the physical dataset values and not from the output of the predictor $H_2$ itself. The errors increase dramatically when the input of $H_2$ is taken from its previous output, yielding an impossibility to use it as an integrator of the dynamical system.

On the other hand, the performance of $H_2$ is comparable to the performance of $H_1$, when comparing the relative errors of both models. From a comparison of the resulting graphical illustrations, $H_2$ seems to perform better on the testing sets.
Figure 8. Graphical diagram of the deep convolution neural network used for the fitting of $H_2$. All convolutions employ “same” padding (also called “half” padding). If the input and output of a convolution have shapes $I_1 \times I_2 \times I_3$ and $O_1 \times O_2 \times O_3$, respectively, then the dimensions are obtained as $O_i = \lceil I_i / s_i \rceil$, where $s_i$ denotes the stride along the dimension $i$, for $i = 1, 2$. The dimension $O_3$ is determined by the number of filters.

Figure 9. Sample results of $H_2$ surrogate model for the training datasets number 10, as shown in Table 1. (a) $A$; (b) $V$; (c) $T_G$; (d) $T_L$. 
Figure 10. Sample results of $\mathcal{H}_2$ surrogate model for the testing datasets number 20, as shown in Table 1. (a) $A$; (b) $V$; (c) $T_G$; (d) $T_L$.

Figure 11. Sample results of $\mathcal{H}_2$ surrogate model for the testing datasets number 22, as shown in Table 1.
According to the previously introduced notation, \( \theta \) stands for linear activation or no activation function.

The surrogate model \( \mathcal{H}_3 \) is built using the network illustrated in Table 6 and represented through the diagram in Figure 12. The input of the network is a 3D tensor of size of 20 \( \times \) 5 \( \times \) 1, for 20 time steps and 5 variables as inputs. The selected optimization algorithm is Adam with the loss function being the mean square errors, modified to add a regularization similar to the DMD regularization [36,37], to perform integration in time without increasing the error significantly. For instance, the loss function \( J \) in this section reads:

\[
J = \sum_{j=1}^{19} \left( \| A_j - \hat{A}_j \| + \| V_j - \hat{V}_j \| + \| T_{Gi} - \hat{T}_{Gi} \| + \| T_{Lj} - \hat{T}_{Lj} \| \right) + \lambda_w \sum_{j=1}^{m} \theta_j^2 + \lambda_b \sum_{j=1}^{p} \kappa_j^2
\]

where \( \theta_j \) are all the weights of the last layer of the network described in Table 6 (the linear layer), and \( \kappa_j \) its biases. \( \lambda_w \) and \( \lambda_b \) are regularization parameters. The available data after reformulation into a sequence of 20 steps has the size of 19 \( \times \) 980 inputs for the training sets and 4 \( \times \) 980 for the testing set. A batch size of 980 is selected with 1500 epochs for the training. The initial learning rate was set to \( \alpha = 10^{-4} \). The input shape has therefore of a size of 20 \( \times \) 5 \( \times \) 1. The results are shown for \( \lambda_w = \lambda_b = 10^{-5} \).

### Table 5. Mean relative errors of the trained incremental model \( \mathcal{H}_2 \) with \( t_s = 4 \).

| Model \( \mathcal{H}_2 \) | \( \mathcal{E}_A \) | \( \mathcal{E}_V \) | \( \mathcal{E}_{T_{Gi}} \) | \( \mathcal{E}_{T_{Lj}} \) |
|--------------------------|-----------------|-----------------|-------------------|-------------------|
| Training sets            | 0.11%           | 0.81%           | 1.91 \times 10^{-5} % | 0.25%             |
| Testing sets             | 0.039%          | 0.898%          | 1.94 \times 10^{-5} % | 0.284%            |

### Table 6. Structure of the deep convolution neural network used for the fitting of \( \mathcal{H}_3 \).

| Layers | Shape | Activation |
|--------|-------|------------|
| 1      | 2D convolution, 30 filters, kernel \((4 \times 5)\), strides \((2 \times 1)\), “same” padding | selu |
| 2      | 2D convolution, 60 filters, kernel \((2 \times 1)\), strides \((2 \times 1)\), “same” padding | selu |
| 3      | 2D convolution, 120 filters, kernel \((5 \times 1)\), strides \((1 \times 1)\), “same” padding | selu |
| 4      | A flatten layer, reshapes all inputs into a single vector | no activation |
Table 6. Cont.

|    | Fully connected dense layer with 1200 neurons | tanh |
|----|---------------------------------------------|------|
| 5  | Fully connected dense layer with 125 neurons | selu |
| 6  | Fully connected dense layer with 70 neurons  | selu |
| 7  | Fully connected dense layer with 25 neurons  | selu |
| 8  | Fully connected dense layer with 4 neurons   | linear |

Figure 12. Graphical diagram of the deep convolution neural network used for the fitting of $H_3$. All convolutions employ “same” padding (also called “half” padding). If the input and output of a convolution have shapes $I_1 \times I_2 \times I_3$ and $O_1 \times O_2 \times O_3$, respectively, then the dimensions are obtained as $O_i = \lfloor I_i / s_i \rfloor$, where $s_i$ denotes the stride along the dimension $i$, for $i = 1, 2$. The dimension $O_3$ is determined by the number of filters.

The results using physical inputs for a selected training dataset 10 are shown in Figure 13, whereas the results for the selected testing datasets 20 and 22 in Figures 14 and 15, respectively. The relative errors are shown in Table 7, computed using Equations (5) and (6), as illustrated in Section 3.1.

Figure 13. Cont.
Figure 13. Sample results of $\mathcal{H}_3$ surrogate model for the training dataset number 10, as shown in Table 1.

Figure 14. Sample results of $\mathcal{H}_3$ surrogate model for the testing datasets number 20, as shown in Table 1.

Table 7. Mean relative errors of the trained incremental model $\mathcal{H}_3$ with $t_s = 20$.  

| Model $\mathcal{H}_3$ | $\mathcal{E}_A$ | $\mathcal{E}_V$ | $\mathcal{E}_{T_G}$ | $\mathcal{E}_{T_L}$ |
|------------------------|-----------------|-----------------|---------------------|---------------------|
| Training sets          | 0.058%          | 1.21%           | $9.18 \times 10^{-6}$% | 0.039% |
| Testing sets           | 0.012%          | 1.4%            | $9.11 \times 10^{-6}$% | 0.04% |
Figure 15. Sample results of $H_3$ surrogate model for the testing datasets number 22, as shown in Table 1.

3.2.3. Integrator Results

The neural network described in Table 6, complemented with the regularized loss given in Equation (9), can be used to integrate in time the system behavior. For instance, we can write

$$\begin{pmatrix} \hat{A}_i \\ \hat{V}_i \\ T_{Gi} \\ T_{Li} \end{pmatrix} = H_3 \begin{pmatrix} \hat{A}_{i-1} \\ \vdots \\ \hat{A}_{i-20} \\ \hat{V}_{i-1} \\ \vdots \\ \hat{V}_{i-20} \\ T_{Gi-1} \\ \vdots \\ T_{Gi-20} \\ T_{Li-1} \\ \vdots \\ T_{Li-20} \end{pmatrix}.$$

(10)

The integrator is stable enough to compile all the time steps. Figure 16 shows the integrator results in a training case, whereas Figures 17 and 18 show the integrator results on two testing cases. The integrator errors are given in Table 8.
Figure 16. Sample integrator results of $\mathcal{H}_3$ surrogate model when integrating the training datasets number 10, as shown in Table 1.

Figure 17. Sample integrator results of $\mathcal{H}_3$ surrogate model when integrating the testing datasets number 20, as shown in Table 1.

Table 8. Mean relative errors of the trained incremental model $\mathcal{H}_3$ with $t_s = 20$, when performing an integrator operation as per Equation (10).

| Model | $\mathcal{H}_3$ | $\mathcal{E}_A$ | $\mathcal{E}_V$ | $\mathcal{E}_{T_G}$ | $\mathcal{E}_{T_L}$ |
|-------|-----------------|-----------------|-----------------|-------------------|-------------------|
|       |                 |                 |                 |                   |                   |
| Training sets |                     | 1.7%            | 1.54%           | $2.07 \times 10^{-5}$% | 0.091%           |
| Testing sets  |                     | 0.44%           | 1.05%           | $2.9 \times 10^{-5}$% | 0.045%           |

It can be clearly noticed that the integrator performs extremely well on the selected datasets.
Figure 18. Sample integrator results of $H_3$ surrogate model when integrating the testing datasets number 22, as shown in Table 1.

4. Conclusions

In this work several prediction models for output time series from input time series have been compared. Without loss of generality, a coupled problem in thermohydraulics has been considered, but the procedure generalizes to many other phenomena and fields. Moreover, instead of simulation-based data, experimental data can be used.

The models are based on different deep neural networks architectures. If a parametrization of the input function is available, a non-incremental approach involving transposed convolution neural networks allows the prediction of the whole sequence, as described and discussed in Section 3.1. The non-incremental approach is based on convolution transpose layers, which takes a low amount of data as an input and increase the size of the output to reach a desired dimension. These layers appears as a suitable approach in this situation where only few inputs are required to built the whole sequence of time measurements.

The incremental models (dynamical system integrators) presented in Section 3.2 are based on regressing the variable on its own lagged (i.e., past) values together with the ones of the input, through standard deep convolution neural networks. Therefore, the prediction occurs step-by-step. The difference between the two incremental models stands in the delay (or lag) factor $t_s$. A small value of $t_s$ allows really fast predictions. However, the model can be unstable when its input is taken from the previous model outputs (this occurs, for instance, when $t_s = 4$). In such case, the surrogate cannot be used as an integrator of the system. Results are thoroughly improved for $t_s = 20$, where also a regularization term was introduced in the loss function. The incremental approach is based on convolution layers. In fact, these layers will take a large input and reduce the input size to extract relevant information. The filters are designed with a large dimension along the time axis, to allow easier extraction of the inputs’ time variation.

The non-incremental approach can be clearly faster to evaluate the output of the solution, however, the integrator clearly outperforms the non-incremental approach. Moreover, the integrator can be used on novel datasets, and potentially outside of the simulated time.
frame. The initialization requires only $t_i$ time-steps to be simulated, and the remaining dynamics is predicted in real time, allowing huge computational gains. However, the non-incremental approach can’t be used on a different time frame, out of the simulated region, as the output is hardly coded to predict only $N_t = 1000$ time steps.

**Author Contributions:** Conceptualization, C.G. and F.C.; methodology, C.G and A.P.; software, C.G.; formal analysis, C.G.; investigation, V.C. and A.P.; data curation, V.C. and A.P.; writing—original draft preparation, C.G.; writing—review and editing, V.C. and A.P.; supervision, F.C.; project administration, F.C.; funding acquisition, F.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data is available upon request.

**Acknowledgments:** The authors greatly acknowledge the support of Anoop Ebey Thomas from ESI Group and all the partners of the PSPC Réacteur Numérique project.

**Conflicts of Interest:** The authors declare no conflict of interest.

**References**

1. Markovsky, I.; Rapisarda, P. Data-driven simulation and control. *Int. J. Control.* 2008, 81, 1946–1959. https://doi.org/10.1080/00207170801942170.

2. Box, G.; Jenkins, G.M. *Time Series Analysis: Forecasting and Control*; Holden-Day, California, USA : 1976.

3. Pollock, D.; Green, R.; Nguyen, T. *A Handbook of Time-Series Analysis, Signal Processing and Dynamics*; Signal Processing and Its Applications; Academic Press: Cambridge, MA, USA, 1999.

4. Kantz, H.; Schreiber, T. *Nonlinear Time Series Analysis*; Cambridge University Press: Cambridge, UK, 2003.

5. Palit, A.K.; Popovic, D. *Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications (Advances in Industrial Control)*; Springer: Berlin/Heidelberg, Germany, 2005.

6. Lian, Y.; Wang, R.; Jones, C.N. Koopman based data-driven predictive control. *arXiv* 2021, arXiv:2102.05122.

7. Klas, S.; Nüske, F.; Peitz, S.; Niemann, J.H.; Clementi, C.; Schütte, C. Data-driven approximation of the Koopman generator: Model reduction, system identification, and control. *Phys. D Nonlinear Phenom.* 2020, 406, 132416. https://doi.org/10.1016/j.physd.2020.132416.

8. Otto, S.; Rowley, C. Koopman Operators for Estimation and Control of Dynamical Systems. *Annu. Rev. Control. Robot. Auton. Syst.* 2021, 4, 59–87. https://doi.org/10.1146/annurev-control-071020-010108.

9. Maksakov, A.; Palis, S. Koopman-based data-driven control for continuous fluidized bed spray granulation with screen-mill-cycle. *J. Process. Control.* 2021, 103, 48–54. https://doi.org/10.1016/j.jprocont.2021.05.006.

10. Williams, M.O.; Kevrekidis, I.G.; Rowley, C.W. A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. *J. Nonlinear Sci.* 2015, 25, 1307–1346. https://doi.org/10.1007/s00332-015-9258-5.

11. Zhang, C.; Zuazua, E. A quantitative analysis of Koopman operator methods for system identification and predictions. *2021, in press.*

12. Tijskens, A.; Roels, S.; Janssen, H. Neural networks for metamodelling the hygrothermal behaviour of building components. *Build. Environ.* 2019, 162, 106282. https://doi.org/10.1016/j.buildenv.2019.106282.

13. Connor, J.; Martin, R.; Atlas, L. Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Netw.* 1994, 5, 240–254. https://doi.org/10.1109/72.279188.

14. Manic, D.; Mandic, D.; Chambers, J. *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*; Adaptive and Cognitive Dynamic Systems: Signal Processing; Wiley: Hoboken, NJ, USA, 2001.

15. Waibel, A.; Hanazawa, T.; Hinton, G.; Shikano, K.; Lang, K. Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoust. Speech, Signal Process.* 1989, 37, 328–339. https://doi.org/10.1109/29.21701.

16. Peddinti, V.; Povey, D.; Khudanpur, S. A time delay neural network architecture for efficient modeling of long temporal contexts. *Proc. Interspeech 2015*, 3214–3218. https://doi.org/10.21437/Interspeech.2015-647.

17. Wähler, C.; Anlauf, J. Real-time object recognition on image sequences with the adaptable time delay neural network algorithm—Applications for autonomous vehicles. *Image Vis. Comput.* 2001, 19, 593–618. https://doi.org/10.1016/S0262-8856(01)00040-3.

18. Pinheiro, E.; Bosco, J.; Lima, L.; Geraldo, F. Nonlinear autoregressive neural networks for forecasting wind speed time series. *Int. J. Dec. Res.* 2021, 10, 40336–40343. https://doi.org/10.37118/ijdjr.19845.09.2020.

19. Ruiz, L.; Cuellar, M.; Calvo-Flores, M.; Pegalajar Jiménez, M.d.C. An Application of Non-Linear Autoregressive Neural Networks to Predict Energy Consumption in Public Buildings. *Energies 2016*, 9, 684. https://doi.org/10.3390/en9090684.

20. Xie, H.; Tang, H.; Liao, Y.H. Time series prediction based on NARX neural networks: An advanced approach. In Proceedings of the 2009 International Conference on Machine Learning and Cybernetics, Hebei, China, 12–15 July 2009; Volume 3, pp. 1275–1279. https://doi.org/10.1109/ICMLC.2009.5212326.
21. Diaconescu, E. The use of NARX neural networks to predict chaotic time series. WSEAS Trans. Comput. Res. 2008, 3, 182–191.
22. Guzman, S.M.; Paz, J.O.; Tagert, M.L.M. The Use of NARX Neural Networks to Forecast Daily Groundwater Levels. Water Resour. Manag. 2017, 31, 1591–1603. https://doi.org/10.1007/s11269-017-1598-5.
23. Menezes, J.M.P.; Barreto, G.A. Long-term time series prediction with the NARX network: An empirical evaluation. Neurocomputing 2008, 71, 3335–3343. https://doi.org/10.1016/j.neucom.2008.01.030.
24. Boussaada, Z.; Curea, O.; Remaci, A.; Camblong, H.; Mrabet Bellaaj, N. A Nonlinear Autoregressive Exogenous (NARX) Neural Network Model for the Prediction of the Daily Direct Solar Radiation. Energies 2018, 11, 620. https://doi.org/10.3390/en11030620.
25. Zeiler, M.D.; Fergus, R. Visualizing and Understanding Convolutional Networks. arXiv 2013, arXiv:1311.2901.
26. Zeiler, M.D.; Taylor, G.W.; Fergus, R. Adaptive deconvolutional networks for mid and high level feature learning. In Proceedings of the 2011 International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; pp. 2018–2025. https://doi.org/10.1109/ICCV.2011.6126474.
27. Bernard, M.; Dellacherie, S.; Faccanoni, G.; Grec, B.; Penel, Y. Study of a low Mach nuclear core model for two-phase flows with phase transition I: Stiffened gas law. ESAIM Math. Model. Numer. Anal. 2014, 48, 1639–1679. https://doi.org/10.1051/m2an/2014015.
28. Brown, J.; Maldonado, G. History of PWR and BWR development. Encyclopedia of Nuclear Energy, Pages 157-171, Elsevier 2021. https://doi.org/10.1016/B978-0-12-819725-7.00149-5.
29. Brown, J.A. Pressurized Water Reactors. In Encyclopedia of Nuclear Energy; Greenspan, E., Ed.; Elsevier: Oxford, UK, 2021; pp. 196–213. https://doi.org/10.1016/B978-0-12-819725-7.00129-X.
30. Liu, Z.T.; Qin, B.K.; Xie, H.; Wang, B.H. Status and trends of thermal-hydraulic system codes for nuclear power plants with pressurized water reactors. Yuanzheneng Kejixue jishu/At. Energy Sci. Technol. 2009, 43, 966–972.
31. Zhang, K. The multiscale thermal-hydraulic simulation for nuclear reactors: A classification of the coupling approaches and a review of the coupled codes. Int. J. Energy Res. 2020, 44, 3295–3315. https://doi.org/10.1002/er.5111.
32. Emonet, P.; Souyri, A.; Gandrille, J.; Barré, F. CATHARE-3: A new system code for thermal-hydraulics in the context of the NEPTUNE project. Nucl. Eng. Des. 2011, 241, 4476–4481.
33. Klambauer, G.; Unterthiner, T.; Mayr, A. Self-Normalizing Neural Networks. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; pp. 1–102.
34. Kingma, D.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, CA, USA, 9–12 June 2013.
35. Sancarlos, A.; Cameron, M.; Le Puwedec, J.M.; Groulier, J.; Duval, J.L.; Cueto, E.; Chinesta, F. Learning stable reduced-order models for hybrid twins. Data-Centric Eng. 2021, 2, e10. https://doi.org/10.1017/dce.2021.16.
36. Erichson, N.; Liu, L.M.; Kutz, J.; Brunton, S. Randomized dynamic mode decomposition. SIAM J. Appl. Dyn. Syst. 2019, 18, 1867–1891.