

Dynamical System Simulation with Attention and Recurrent Neural Networks

Javier Fañanás-Anaya , Gonzalo López-Nicolás , Carlos Sagüés 

Instituto de Investigación en Ingeniería de Aragón (I3A),

Universidad de Zaragoza, Spain

javierfa@unizar.es, gonlopez@unizar.es, csagues@unizar.es

Abstract—Accurate and efficient real-time simulation of nonlinear dynamic systems remains an important challenge in fields such as robotics, control systems and industrial processes, requiring innovative solutions for predictive modeling. In this work, we introduce a novel recurrent neural network (RNN) architecture designed to simulate complex nonlinear dynamical systems. Our approach aims to predict system behavior at any time step and over any prediction horizon, using only the system's initial state and external inputs. The proposed architecture combines RNN with multilayer perceptron (MLP), and incorporates an attention mechanism to process both previous state estimates and external inputs. By training without teacher forcing, our model can iteratively estimate the system's state over long prediction horizons. Experimental results on three public benchmarks show that our method outperforms other state-of-the-art solutions. We highlight the potential of our proposal for modeling and simulating nonlinear systems in real-world applications.

Index Terms—Dynamical System, Recurrent Neural Network, Attention, Hybrid Architecture, Real-Time Simulation, Training Algorithm

I. INTRODUCTION

Simulation of dynamical systems in real-time is necessary for various applications across different domains such as robotics [1], [2], control systems [3], [4], finance [5], [6], weather forecasting [7], [8], traffic forecasting [9], [10] and health [11], [12], among others. The ability to accurately predict the behavior of these systems enables proactive decision-making and effective control strategies. Traditional methods often rely on mathematical models, which may become impractical or inaccurate in complex or dynamic environments.

Although traditional mathematical models are widely used, they are often based on simplifying assumptions that may not account for certain dynamics, resulting in unmodeled behaviors. While these discrepancies may produce negligible errors in short-term predictions, they can significantly affect the accuracy of long-term forecasts. Consequently, there has been a growing interest in leveraging neural networks (NN) to simulate dynamical systems because of their ability to learn complex patterns from data or to use these architectures as a surrogate for computationally expensive physical models [13].

Among all types of NN, the use of recurrent neural networks (RNN) stands out in the simulation of dynamical systems. In the case of the control engineering, there are numerous works that use different architectures with RNN in model predictive control schemes. For example, Long Short-Term Memory

(LSTM) as a model of the motor [14], outperforms classical approaches in simulating nonlinear dynamical systems. Another approach uses an LSTM architecture as a predictor in a control scheme [15], in this case applied to a vehicle velocity and position prediction problem. RNN are also used for water level prediction [16]. These works show that RNN can improve performance in control schemes with respect to classical approaches. However, these works are focused on the integration of these RNN in the control schemes and not on minimizing the error of these architectures with respect to real systems.

In robotics, there is also interest to simulate real-time dynamical systems, where using RNN architectures can also obtain good results. As a first example, we highlight the integration of a hybrid architecture with RNN and Feedforward NN in a control scheme of a robotic arm tasked to cut fruits and vegetables [17]. In that work, however, the proposed architecture is a one time step predictor as instabilities and errors were found when it is used with a larger prediction horizon. Another work shows how to model and control a small helicopter using RNN [18], however, in robotics, the trend in research is to use hybrid architectures where NN are combined with physical models with knowledge of the system. A hybrid model of classical identification techniques and convolutional neural networks (CNN) models a helicopter in [19], outperforming previous work in this domain. Using LSTM architectures also works well for predicting the behavior of a quadrotor for a short prediction horizon of two seconds [20]. A hybrid alternative is also presented in that paper, in which physical models are included to improve the overall accuracy of the system. The prediction of the behavior of the same quadrotor was improved for a prediction horizon of less than one second, by using a hybrid architecture of NN and physical models [21]. In that example, CNN are used instead of RNN, limiting in that case the solution to a sequence-to-sequence approach. The potential of using NN as surrogate models to simulate dynamical systems is shown in [22]. In that paper, LSTM, hybrid CNN-LSTM architectures and a Gaussian process regression organized in a nonlinear autoregressive exogenous architecture (NARX) are compared. In the experiments of that work, the NARX solution is shown to be more stable and generally performs better than the other proposed NN. A hybrid CNN-LSTM-MLP architecture for one-hour-ahead solar irradiance forecasting is presented in [24]. The hybrid architecture outperforms the alternatives

TABLE I: Summary of relevant articles related to the simulation of dynamic systems using mainly NN-based techniques.

Reference	Application	Architecture	Type	Horizon
[14]–[16]	Model Predictive Control	LSTM	Data-Driven	Fixed
[17]	Robotic Model and Control	RNN-MLP	Data-Driven	Fixed
[18]	Helicopter Model and Control	RNN	Data-Driven	Fixed
[19]	Helicopter Model and Control	CNN	Hybrid Model	Fixed
[20]	Quadrotor Multistep Prediction	LSTM	Hybrid Model	Fixed
[21]	Quadrotor Multistep Prediction	CNN	Hybrid Model	Fixed
[22]	Dynamical Systems Simulation	NARX-NN	Data-Driven	Fixed
[23]	Dynamical Systems Simulation	MLP-CNN	Data-Driven	Fixed
[24]	Solar Irradiance Forecasting	CNN-LSTM-MLP	Data-Driven	Fixed
[25]	Flood Forecasting	Attention -LSTM	Data-Driven	Fixed
[26]	Time Series Forecasting	Attention -LSTM	Data-Driven	Fixed
[27], [28]	State Estimation of Batteries	Attention -LSTM	Data-Driven	Fixed
[29]	System Identification With Outliers	RFS-LSSVR	Data-Driven	Any (Test)
[30]	System Identification	S-REVARB	Data-Driven	Any (Test)
[31]	System Identification With Outliers	LMN	Data-Driven	Any (Test)
Ours	Dynamical Systems Simulation	MA -LSTM-MLP	Data-Driven	Any (Train & Test)

that use only MLP or RNN. RNN not only have potential in modeling dynamical systems, but can also be effectively used in parameter identification problems [32]. Another work compares the use of MLP and RNN to identify complex nonlinear systems [23], finding that the best approach is to use a hybrid solution with both architectures.

Although the NN structure of attention mechanisms can be found in many applications, exploration is needed to effectively include them in architectures for simulating dynamical systems. These architectures are present in problems related to sequence-to-sequence predictions or time series forecasting. We highlight the work of [25], which proposes a hybrid architecture that uses attention with LSTM for flood forecasting. The proposed architecture outperforms approaches that use only MLP, LSTM or CNN. The limitation of the Attention-LSTM architecture proposed in that paper is a sequence-to-sequence approach, i.e., it predicts the flood for a fixed window of time from another fixed-size window of observation. This approach, although accurate for many scenarios, limits using these architectures in real time and adapting to unexpected changes, which is important when simulating a dynamical system with external inputs. Another example of sequence-to-sequence time series prediction is [26], where the proposed evolutionary attention LSTM improves the results of using basic LSTM. A hybrid LSTM architecture with self-attention for state estimation of lithium-ion batteries is used in [27]. That example can be considered a dynamical system, controlled by external inputs. The cited example predicts the state of the battery for a fixed period of time, knowing the external parameters for that period of time, leading the solution to a sequence-to-sequence problem. The self-attention is applied to the hidden states calculated by the LSTM layer, obtaining better performance than using only LSTM layers. Another work that also proposes a hybrid architecture of attention and LSTM for state estimation of lithium-ion batteries is [28], with the difference that in this case the self-attention layer is positioned before the LSTM layer. The work of [33] presents a solution that estimates attitude and position of shield machine in tunneling from multiple inputs using a hybrid LSTM-Attention architecture, again improving the results of using only LSTM. These examples show how the capacity of RNN

can be improved using attention mechanisms. These results motivate the development of these architectures for real-time dynamical systems simulation problems.

In addition to the application of RNN for system identification and modeling, other techniques are present in the state of the art. Two novel least squares support vector regression (LSSVR) models are proposed in [29], called Robust Fixed-Size LSSVR (RFS-LSSVR) and Reweighted Robust Fixed-Size LSSVR (R^2 FS-LSSVR). One of the main novelties is that both architectures are robust to the presence of outliers in the external inputs. Another example uses a Stochastic Recurrent Variational Bayes (S-REVARB) framework for identification and modeling of dynamical systems [30]. Local model network (LMN) for the identification and modeling of recursive dynamical systems has been studied in [31], again showing robustness to outliers in the system inputs. These last three examples have been tested in scenarios without a fixed prediction horizon, also called free simulation, showing good results. Although these architectures are trained in scenarios where there is no feedback from the output to the input of the next time step, they have shown good performance in free simulation, where there is feedback.

State-of-the-art approaches using NN to simulate dynamical systems have certain limitations that motivate the need for further developments. Some important challenges are:

- **Fixed prediction horizons:** State-of-the-art approaches usually operate with fixed forecast horizons, which limits their adaptability to different scenarios. Working with fixed prediction windows leads to sequence-to-sequence approaches, which present limitations for real-time applications. This makes it difficult to use these architectures to monitor the system in real time or to easily adapt to unexpected changes.
- **Hybrid physical architectures:** While hybrid architectures incorporating physical models and NN have shown success, they might not be universally applicable, especially when the underlying physics of a system is unknown.
- **Novel NN approaches:** State-of-the-art solutions are limited to working with different RNN structures. Attention is present in the state of the art of many applications,

especially where data sequences are involved. Simulation of dynamical systems presents sequences of data, and incorporating attention mechanisms into the architectures has not been explored.

Table I shows a summary of relevant articles related to the simulation of dynamic systems using mainly NN-based techniques. In addition to the references, it shows the application domain, the architecture used, whether the method is data-driven or a hybrid of a physical model with NN (Hybrid Model), as well as if the proposal has been tested in scenarios with or without a fixed prediction horizon. Note that in the architecture category the use of Attention or Multi-Head Attention (MA) is highlighted.

In view of the review of the state of the art, and the challenges presented, the research gaps that this paper aims to fill are: There is no NN architecture based on the combination of attention, RNN and MLP techniques. Data-driven architectures are needed to facilitate generalization to any dynamic system for which data is available. In addition, applications require architectures that operate without a prediction horizon in the testing and training phases, training themselves with feedback from their own predictions with a custom algorithm.

The context of this paper is the simulation of dynamical systems with external inputs, where only the initial state of the system at any time step is known, and external inputs can be received in real time or from a time sequence of any length. The challenge is to develop an architecture that can handle any prediction horizon, allowing both short-term and long-term simulations. In addition, the architecture must be able to operate in real time, adapting to external inputs as they are received. Our main contributions are:

- We propose an architecture that runs iteratively, allowing the dynamical system to be predicted for any prediction horizon when external inputs are known. Moreover, the architecture adapt to real-time scenarios, taking inputs dynamically to provide on-line predictions. To make this approach work, a customized algorithm based on training without teacher forcing has been implemented.
- Our proposed hybrid architecture, called MA-LSTM-MLP, involves LSTM and MLP with attention mechanisms. This architecture integrates attention mechanisms to exploit historical information on external inputs and system state estimations. This hybrid architecture outperforms the same architecture without attention.
- Our architecture works as a black box, eliminating the need for detailed knowledge of the underlying physical system. This feature allows its application to any system for which we can have a database with the evolution of its states and the external inputs applied.
- We evaluate the performance of the proposed architecture using public benchmarks for nonlinear dynamical systems with external inputs, comparing the results with state-of-the-art approaches.

II. PRELIMINARIES

This section defines the problem of simulating dynamic systems and presents the minimal RNN architecture that meets

the basic requirements. Consider the state of a dynamical system at a time instant t by $y_t \in \mathbb{R}^m$, and the external inputs at t by $u_t \in \mathbb{R}^n$. Consider a function f that estimates the state of the dynamical system at the next time instant $\hat{y}_{t+1}$ from u_t and $\hat{y}_t$:

$$\hat{y}_{t+1} = f(u_t, \hat{y}_t) \quad (1)$$

Simulating dynamical systems aims to calculate over a time period T its state evolution $Y_{1,T} \in \mathbb{R}^{m \times T}$

$$Y_{1,T} = [y_1, y_2, \dots, y_T]. \quad (2)$$

Knowing y_0 , and the external inputs over the entire time period $U_{0,T-1} \in \mathbb{R}^{n \times T}$

$$U_{0,T-1} = [u_0, u_1, \dots, u_{T-1}], \quad (3)$$

function f can be iterated T times until getting an estimate $\hat{Y}_{1,T} \in \mathbb{R}^{m \times T}$

$$\hat{Y}_{1,T} = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_T]. \quad (4)$$

Note that y_0 can be the state of the dynamical system at any instant of its execution, this will be the known initial state from which our simulation will start. In the same way, T will depend on how many future external inputs are known, allowing to adapt any value. The main problem to be solved in this paper is to obtain a function f using RNN that minimizes the error between $Y_{1,T}$ and $\hat{Y}_{1,T}$.

There are several reasons to seek a function that calculates only the state of the system at the next time instant and follows an iterative approach to simulate the behavior of the system over time. This is in contrast to employing a function for multistep prediction, where the time length of the external inputs and the prediction horizon is fixed.

- **Generalization to any prediction horizon:** An iterative approach allows for flexibility in predicting the system's behavior over different time horizons. By adjusting the number of iterations, the architecture can be applied to short-term or long-term simulations, providing a versatile solution for various forecasting needs or depending on the number of known external inputs.
- **Real-time applications:** This approach allows to simulate the behavior of the dynamical system by receiving the external inputs in real time, which allows to monitor the state of the system and to adapt to unexpected changes.
- **Reduced computational complexity:** Estimating states iteratively allows computationally less demanding architectures. This approach significantly impacts memory efficiency, particularly with longer prediction horizons, avoiding the need to load all external inputs into memory at once or calculate all states simultaneously.

Now we will describe the minimal NN architecture that meets the requirements of this paper. Feedforward NN have been widely used in control problems of dynamical systems [34]–[37]. However, these networks lack memory and cannot model dynamics on their own, so they cannot be used to simulate dynamical systems, they can only be used as single-step predictors [38], [39]. A NN structure suitable for the problem presented in this paper is Nonlinear AutoRegressive with eXogenous inputs Neural Network (NARX-NN) [40],

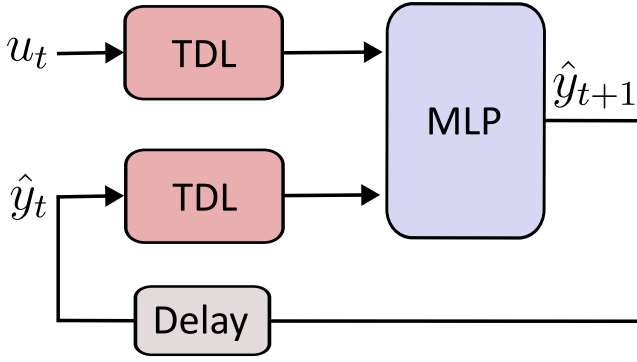


Fig. 1: Diagram of the NARX-NN architecture. The architecture uses a MLP to estimate $\hat{y}_{t+1}$ from k past external inputs u and k past state estimates $\hat{y}$.

[41], since it has buffers to store information and is a structure with recurrent connections to model dynamics.

In NARX-NN there is a nonlinear function f that estimates the next system state $\hat{y}_{t+1}$ from buffers that store the k previous system states $\hat{Y}_{t-k,t}$ and the k past external inputs $U_{t-k,t}$:

$$\hat{y}_{t+1} = f(u_t, u_{t-1}, \dots, u_{t-k}, \hat{y}_t, \hat{y}_{t-1}, \dots, \hat{y}_{t-k}) \quad (5)$$

This architecture consists of these buffers, called tapped delay lines (TDL), and a multilayer perceptron (MLP) with a recurrent connection between the network output and input [42]. For a MLP with L layers, where $l = 1, \dots, L$, the output of a layer l at time step t is represented by x_t^l , computed as follows:

$$x_t^l = \sigma(W^l x_t^{l-1} + b^l) \quad (6)$$

where W^l and b^l are the weight matrix and the bias vector of the l -layer, respectively. σ is the activation function applied element-wise. x_t^0 is the concatenation of $\hat{Y}_{t-k,t}$ and $U_{t-k,t}$. Note that $\hat{y}_{t+1} = x_t^L$.

This architecture, although it is the simplest of those to be shown in this paper, is present in the state of the art of simulation of different dynamical systems [43]–[46]. In this paper, it is the baseline to compare with our proposed architecture. A diagram of the NARX-NN is shown in Fig. 1.

III. PROPOSED ARCHITECTURE

This section describes the NN architecture and training algorithm proposed for the simulation of dynamical systems. The implemented architecture fulfill the requirements described in this paper. Their function is to simulate dynamical systems with external inputs, iteratively to enable their use in real time and for any prediction horizon.

A. LSTM-MLP

Before presenting the complete proposed architecture, in this subsection we will introduce the architecture without the attention structure. In this way, we will introduce all the parts

of the hybrid solution so we can compare the final version with this intermediate architecture.

RNN are ideal architectures for representing dynamical systems. These architectures are designed to work with sequences of data, allowing to capture temporal dependencies with their hidden states h and use these to estimate the future states of the dynamical system. RNN consist mainly of two steps: Hidden state update h_t and output generation $\hat{y}_{t+1}$:

$$h_t = f(h_{t-1}, x_t) \quad (7)$$

$$\hat{y}_{t+1} = g(h_t, x_t) \quad (8)$$

where in our context, with a single hidden layer and without TDL buffers, x_t is the concatenation of u_t and $\hat{y}_t$, h_t and $\hat{y}_t$ are the hidden state of the RNN and the estimated state of the dynamical system respectively. The functions f and g will depend on the chosen RNN architecture.

Traditional RNN suffer from the vanishing and exploding gradient problems, limiting their ability to capture long-term dependencies in sequences. Therefore, more complex architectures were introduced to avoid this problem, such as LSTM [47]. LSTM not only uses hidden states, but it also incorporates memory cells c , allowing them to capture and remember information over long sequences. The LSTM architecture involves updating c and h using a set of gates, including an input gate, forget gate, and output gate. These gates regulate the information flow, enabling LSTMs to selectively remember or forget information. With a single hidden layer and without TDL buffers, the LSTM equations in our context are as follows:

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) \\ f_t &= \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) \\ o_t &= \sigma(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) \\ \tilde{c}_t &= \tanh(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \quad (9)$$

where i_t , f_t and o_t are the input gate, forget gate and output gate at time instant t . $\tilde{c}_t$ and c_t are candidate cell state and cell state respectively, and h_t is the hidden state. σ represents the sigmoid activation function, and $\tanh$ is the hyperbolic tangent activation function. W and b are weight matrices and bias vectors, respectively. $\odot$ represents the element-wise product. Note that h_t is also the output of a LSTM cell.

There are two approaches to handle hidden states in LSTMs: stateful and stateless. In the stateless approach, after each use of the NN, the hidden states are reset. While in the case of the stateful approach, we control when these hidden states are restarted. Using stateless LSTM has benefits when we know the prediction horizon, as it allows us to fix this parameter and to parallelize and speed up the training. In our case, since we use the NN iteratively, we generate a single output in each iteration, so we want the hidden state of the LSTM layers to be kept and updated for the duration of the simulation of the dynamical system. This is why we decided to work with stateful LSTM.

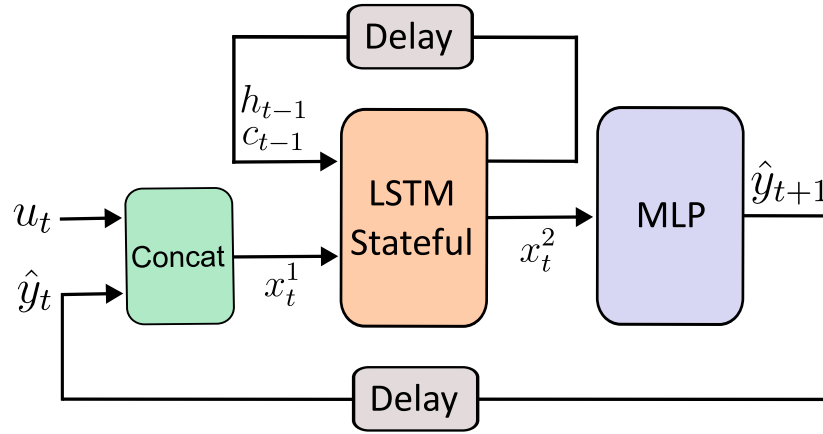


Fig. 2: Schematic of the hybrid LSTM-MLP architecture. At an instant t , the input are the external inputs u_t and the past state estimation $\hat{y}_t$. These two vectors are stacked into x_t^1 , and this is the input of the LSTM stateful layer along with the hidden states from the previous iteration h_{t-1} and c_{t-1} . The LSTM layer captures long-term dependencies and relevant information. The output of the LSTM layer, x_t^2 , is the input of the MLP. Note that $x_t^2 = h_t$. The MLP output is the state of the dynamical system at the next time instant $\hat{y}_{t+1}$.

During the experimentation we observed that in our context using only LSTM layers works well, but it is possible to improve the accuracy and reduce the number of parameters with a hybrid LSTM-MLP approach (Fig. 2), with a single LSTM layer. The intuition behind this hybrid approach is that the LSTM layer captures long-term dependencies and information, while the MLP is responsible for interpreting the non-linearities of the dynamical system and predicting the next time instant.

At an instant t in the LSTM-MLP architecture, the inputs are u_t and $\hat{y}_t$. These two vectors are stacked into x_t^1 , and this is the input of the LSTM stateful layer along with the hidden states from the previous iteration h_{t-1} and c_{t-1} . The output of the LSTM layer, x_t^2 , is the input of the MLP. Note that $x_t^2 = h_t$. The MLP output is the state of the dynamical system at the next time instant $\hat{y}_{t+1}$.

B. MA-LSTM-MLP

Using TDLs in the input of the NARX-NN architecture is necessary to obtain good performance, since it is the memory mechanism. In the case of the LSTM-MLP architecture, it is not necessary to include TDLs to obtain good performance since the LSTM layer incorporates memory. However, the prediction capability of RNN can be improved by including these buffers [20].

In our experiments, we have observed that it is difficult to define the appropriate size of these TDLs, observing that often the NN struggles to effectively utilize the information from TDLs. The effectiveness of TDLs also can vary depending on the specific characteristics of the dynamical system being simulated. In order to effectively utilize the information from TDLs regardless of its size, we propose to include a Multi-Head Attention layer in the architecture to improve the ability to capture relevant temporal dependencies.

Attention mechanisms have gained significant importance in diverse domains, revolutionizing NN architectures. Originally introduced in the context of natural language processing (NLP)

[48], attention mechanisms, notably popularized by their incorporation into the transformer architecture, have since then found applications in diverse fields, ranging from computer vision to time series forecasting [49]–[51]. The fundamental idea behind attention is to allow NN to dynamically focus on specific elements within a sequence, assigning varying degrees of importance to each element based on its relevance to the task on which the architecture has been trained.

The attention mechanism involves three key components: queries Q , keys K , and values V . Consider $x_t \in \mathbb{R}^{(n+m)}$ as a vector with information at time t . Given a sequence of vectors $X_{1,T} \in \mathbb{R}^{(n+m) \times T}$ of length T

$$X_{1,T} = [x_1, x_2, \dots, x_T], \quad (10)$$

the transformation of these vectors into $Q \in \mathbb{R}^{d_q \times T}$, $K \in \mathbb{R}^{d_k \times T}$, and $V \in \mathbb{R}^{d_v \times T}$ involves linear projections:

$$\begin{aligned} Q &= W_q X \\ K &= W_k X \\ V &= W_v X \end{aligned} \quad (11)$$

where $W_q \in \mathbb{R}^{d_q \times (n+m)}$, $W_k \in \mathbb{R}^{d_k \times (n+m)}$, and $W_v \in \mathbb{R}^{d_v \times (n+m)}$ are weight matrices. d_q , d_k and d_v are the dimension vectors of query, key and value respectively. In our implementation they are a hyperparameter to be optimized, chosen to be the same value to maintain consistency:

$$d_q = d_k = d_v \quad (12)$$

Once Q , K , and V are obtained, attention is computed as a weighted sum of the values, where the weights are determined by the compatibility of queries and keys:

$$A = \text{softmax} \left(\frac{Q^T K}{\sqrt{d_k}} \right) \quad (13)$$

where $A \in \mathbb{R}^{T \times T}$ is the attention matrix. Each row contains the relative importance of the other states with respect to that

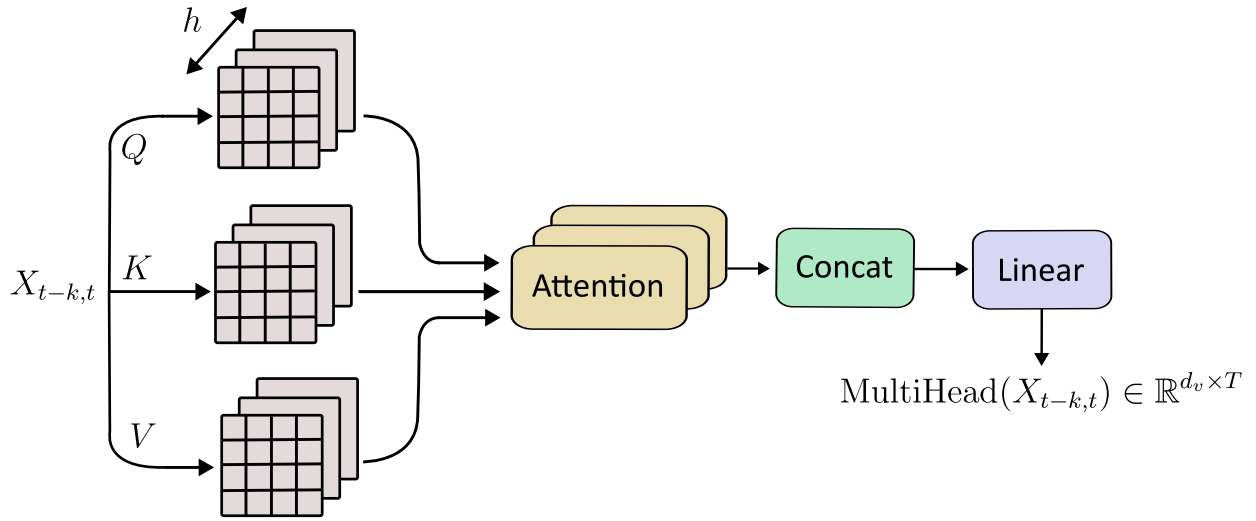


Fig. 3: Diagram of the Multi-Head Attention layer in our context, where $X_{t-k,t}$ represents the TDL buffers at time t . The output is the attention matrix, $\text{MultiHead}(X_{t-k,t})$, which dynamically adjusts the influence of each time step based on its relevance to the simulation task.

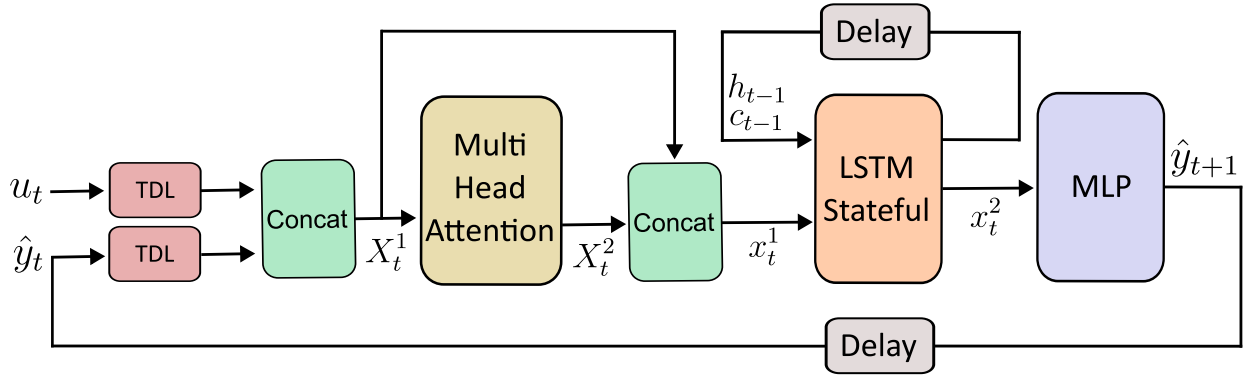


Fig. 4: Schematic of the MA-LSTM-MLP architecture. The Multi-Head Attention layer (Fig. 3) allows to dynamically focus on specific elements within TDL buffers.

specific state. Finally, we calculate the Attention $\in \mathbb{R}^{d_v \times T}$ by weighting the matrix A with V :

$$\text{Attention}(Q, K, V) = VA \quad (14)$$

Note that we derived Q , K , and V from the same sequence X . This attention approach is referred as self-attention, allowing the architecture to weigh the importance of different elements within the same input sequence.

In our architecture we have used Multi-Head Attention. Multi-Head Attention, also introduced in [48], is an extension of the attention mechanism that operates with h sets of queries, keys, and values:

$$\text{MultiHead}(X) = W_o \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h) \quad (15)$$

where $\text{head}_i = \text{Attention}(W_{qi}X, W_{ki}X, W_{vi}X)$

where $W_{qi} \in \mathbb{R}^{d_{qi} \times (n+m)}$, $W_{ki} \in \mathbb{R}^{d_{ki} \times (n+m)}$ and $W_{vi} \in \mathbb{R}^{d_{vi} \times (n+m)}$ are weight matrices of the i -th head. For simplicity, $d_{qi} = d_q/h$, $d_{ki} = d_k/h$ and $d_{vi} = d_v/h$. The output of the $\text{MultiHead} \in \mathbb{R}^{d_v \times T}$ is obtained concatenating the results of each head and linearly projecting them using

$W_o \in \mathbb{R}^{d_v \times h \cdot d_{vi}}$. The idea of Multi-Head Attention is to allow the architecture to pay attention different parts of the input sequence with respect to distinct aspects, enhancing its capacity to capture diverse relationships within the data.

The intuition behind using attention in our context is that it allows the architecture to dynamically focus on specific elements within the TDL, assigning different levels of importance to each element during the prediction process. This adaptive weighting of information proves to be beneficial in scenarios where the importance of past states and inputs varies over time. As a result, the attention layer ensures a context-aware utilization of the TDL information, leading to improved accuracy and robustness across diverse dynamical systems.

In the context of our simulation problem, the input sequence of vectors $X_{t-k,k} \in \mathbb{R}^{(n+m) \times k}$ represents the TDL buffers at time t , which includes both k past system states estimations $\hat{y}$ and external inputs u :

$$X_{t-k,t} = [x_{t-k}, x_{t-k+1}, \dots, x_t], \quad (16)$$

where $x_i = \text{Concat}(u_i, \hat{y}_i)$.

Once the Multi-Head Attention layer that processes the TDL buffers has been defined, we can integrate this layer with the LSTM-MLP to create the MA-LSTM-MLP architecture. Fig. 3 shows a schematic of the Multi-Head Attention layer, where the input is $X_{t-k,t}$ and generates the output attention matrix. A schematic of the complete proposed architecture is shown in Fig. 4. In the schematic, the input matrix for the Multi-Head Attention layer is shown as X_t^1 . Note that $X_t^1 = X_{t-k,t}$. The attention layer output matrix $X_t^2 = \text{MultiHead}(X_{t-k,t})$ is concatenated with the original input matrix X_t^1 and transformed into a 1D feature vector x_t^1 , which will be the LSTM layer input. Experimentation showed that combining the original information with the output of the attention layer improved the architecture's predictions. The architecture continues to run as the LSTM-MLP, detailed in Section III-A.

C. Training algorithm

The NN architecture described in the previous section, as well as the baseline model (Section II), can be considered RNN in which each iteration depends on the prediction of the previous time step. To train these architectures with this iterative approach, there are two possibilities: Train with teacher forcing (TF) or train without teacher forcing (no-TF) [52]. This is also known in the literature as Series-Parallel and Parallel [42]. The TF training approach involves using the true system states from the previous time step during training. The equation for updating the system states during training with TF is as follows:

$$\hat{y}_{t+1} = f(u_t, y_t) \quad (17)$$

where true system state y_t is used. This approach simplifies training as it allows the architecture to learn the relationships between inputs and outputs directly.

In the no-TF training approach, the architecture is trained with its own predictions from the previous time steps $\hat{y}_t$. The training equation aligns with the simulation equation used during inference (Equation 1).

Training with TF has some advantages, the main one being its shorter training time and simplicity. However, architectures are actually trained for single-step predictions, and their inference error increases greatly with long prediction horizons. TF and no-TF training has been studied in chaotic time series prediction [53], [54]. These can be considered as dynamical systems, but without external inputs, since the evolution of the system depends only on the initial conditions. In these cases, the benefit of training with no-TF has been proven. Training with no-TF is more challenging and computationally demanding, as the architecture needs to be iteratively updated based on its own predictions. However, this approach aligns the training process more closely with the inference scenario, making the architecture more robust for longer prediction horizons.

In our experiments, we observed that no-TF training resulted in better accuracy and robustness in the proposed RNN architecture for simulating dynamical systems with external inputs, particularly over longer prediction horizons.

TABLE II: Hyperparameters of our RNN architectures, including those related to training.

Symbol	Hyperparameter	Type
k	TDL buffer size	Architecture
L	Number of hidden layers in the MLP	Architecture
σ	Activation function in the MLP	Architecture
H_M	Neurons per hidden layer in the MLP	Architecture
H_L	Cells in the LSTM layer	Architecture
h	Number of heads in the Attention layer	Architecture
d_k	Key vector dimension in the Attention layer	Architecture
e	Number of epochs	Training
B	Batch size	Training
η	Learning rate	Training
J	Cost function	Training
s	Early stop	Training

D. Architecture hyperparameters

In the previous sections, two RNN architectures have been presented: the NARX-NN, from the state of the art (Section II), and our proposed architecture, the MA-LSTM-MLP (Section III-B). These architectures, together with the training algorithm, present a series of hyperparameters to be optimised during the NN training process. The hyperparameters used are detailed in Table II, specifying if they are parameters corresponding to the architecture or to the training algorithm.

In our MA-LSTM-MLP architecture, as well as in its variation without the attention layer, only one LSTM layer is included, as experimentation showed that adding more layers did not improve performance. The activation functions of the LSTM layer, as detailed in Section III-A, are the sigmoid and hyperbolic tangent activation functions. Other activation functions were tested with worse results.

In the attention layer only the number of heads and the key dimension are defined. The query and value dimension will depend on the input vector. Training with batches and parallelisation is complicated when training with no-TF and stateful RNN. However, we can parallelise the training by processing a number of different runs of a dynamical system at the same time, as long as a prediction horizon is fixed during training. Batch size B represents the number of runs that are processed in parallel during training.

Implementing the proposed architecture, as well as the baseline, with the constraints and requirements stated in this paper such as the combination of different types of layers and the iterative approach, is not trivial. The architectures have been developed using Keras [55] with Tensorflow [56]. The training algorithm must also be implemented with Tensorflow, as there are no functions that allow us to train with no-TF by default. In addition, the design of the architecture and the training algorithm must support the proposed batch training, allowing several systems to be simulated simultaneously.

IV. RESULTS AND DISCUSSION

In this section, the results of the proposed RNN applied to public datasets of nonlinear dynamical systems is presented. Furthermore, qualitative and quantitative comparisons between the results obtained on the test data and the state-of-the-art solutions are provided.

A. Metrics

This section defines the metrics to be used to evaluate the performance of the architectures during training and testing. The objective of the metrics is to evaluate the predictions of the architecture during N time instants $\hat{Y}_{1,N}$ with respect to the real values $Y_{1,N}$. The chosen metrics are Mean Square Error (MSE), Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Normalized Root Mean Square Error (NRMSE), R^2 and Cumulative Absolute Error (CAE).

MSE is a metric typically used as a loss function in NN training. The problem with this metric is its interpretability, as it does not respect the original units and is sensitive to outliers.

$$\text{MSE}(Y_{1,N}, \hat{Y}_{1,N}) = \frac{1}{m \cdot N} \sum_{j=1}^m \sum_{i=1}^N (y_i^j - \hat{y}_i^j)^2 \quad (18)$$

MAE is another commonly used metric that measures the average magnitude of the errors in a set of predictions. It is more interpretable in terms of the original units of the data and is less sensitive to outliers compared to MSE.

$$\text{MAE}(Y_{1,N}, \hat{Y}_{1,N}) = \frac{1}{m \cdot N} \sum_{j=1}^m \sum_{i=1}^N |y_i^j - \hat{y}_i^j| \quad (19)$$

Again, RMSE is an easier metric to interpret, as it respects the original units of the data and is less sensitive to outliers.

$$\text{RMSE}(Y_{1,N}, \hat{Y}_{1,N}) = \sqrt{\frac{1}{m \cdot N} \sum_{j=1}^m \sum_{i=1}^N (y_i^j - \hat{y}_i^j)^2} \quad (20)$$

R^2 measures the proportion of the variance in the true values $Y_{1,N}$ that capture the predictions $\hat{Y}_{1,N}$ of the architecture. It ranges from 0 to 1, where 1 indicates a perfect fit.

$$R^2(Y_{1,N}, \hat{Y}_{1,N}) = 1 - \frac{\sum_{j=1}^m \sum_{i=1}^N (y_i^j - \hat{y}_i^j)^2}{\sum_{j=1}^m \sum_{i=1}^N (y_i^j - \bar{Y}_{1,N}^j)^2} \quad (21)$$

Note that $\bar{Y}_{1,N}^j$ is the mean of parameter j of the N true samples:

$$\bar{Y}_{1,N}^j = \frac{1}{N} \sum_{i=1}^N y_i^j \quad (22)$$

The NRMSE considers the errors relative to the standard deviation of the true values. It provides a normalized measure of the model's performance, making it easier to compare across different datasets.

$$\text{NRMSE}(Y_{1,N}, \hat{Y}_{1,N}) = \sqrt{\frac{1}{m \cdot N} \sum_{j=1}^m \sum_{i=1}^N \left(\frac{y_i^j - \hat{y}_i^j}{\sigma_{1,N}^j} \right)^2} \quad (23)$$

Note that the standard deviation $\sigma_{1,N}^j$ is calculated for each parameter j over the N time instants:

$$\sigma_{1,N}^j = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i^j - \bar{Y}_{1,N}^j)^2} \quad (24)$$

CAE is a metric that evaluates the overall accuracy of a prediction. It is especially useful if it is calculated for all time steps of the prediction horizon, which allows to observe the error stability of the evaluated architecture.

$$\text{CAE}(Y_{1,N}, \hat{Y}_{1,N}) = \frac{1}{m} \sum_{j=1}^m \sum_{i=1}^N |y_i^j - \hat{y}_i^j| \quad (25)$$

All the metrics presented obtain a single value for the m parameters of a dynamical system. We can also use these metrics individually for each parameter of the system, where $m = 1$.

B. Wiener-Hammerstein Process Noise System

The Wiener-Hammerstein Process Noise System dataset [59] consists of a collection of one-dimensional input-output data from an electrical circuit. The complexity of the system consists of a Wiener-Hammerstein model, characterised by the presence of a non-linearity between two linear dynamical systems. Furthermore, noise is introduced before the nonlinear system, complicating the challenges associated with system identification and modeling. The noise is a filtered white Gaussian noise sequence. The filtered noise is generated starting from a discrete-time 3rd order low-pass Butterworth filter followed by a zero-order hold reconstruction and an analog low-pass reconstruction filter with a cut-off frequency of 20 kHz.

The test data consists of two executions of the system during 16,384 time steps, one execution of the system with a multisine input and the other with a swept sine input. The training data consists of 200 realizations, where in each run the execution time varies from 4,096 to 65,536 time steps. During the training phase, a prediction horizon of 2,048 time steps has been set. Having a fixed prediction horizon during training helps to parallelize the training. A 10% of the training data was used as validation.

An effort has been made to make a fair comparison between the baseline and the proposed architecture. Therefore, the architectures have been trained with the same hyperparameters related to the training, and the number of parameters of the architectures are in the same order of magnitude. These training hyperparameters were selected through a grid search (Table III). Adam has been chosen as the optimizer [60].

The strategy for the hyperparameters related to the architecture was as follows. A gridsearch was performed for the NARX-NN architecture. For the case of the LSTM-MLP architecture, we found the best performance by keeping the hyperparameters of the NARX-NN architecture for the MLP, reducing L to 2, and adding an LSTM layer. For the LSTM-MLP architecture, we studied the variants of including TDL with $k = 10$ and using the architecture without TDL. For the MA-LSTM-MLP architecture, the optimal configuration was found by starting from the LSTM-MLP structure with

TABLE III: Hyperparameters of the three architectures and their training configurations for the Wiener-Hammerstein Benchmark, including the specific values explored during the grid search.

Architecture	Hyperparameter	Value	Search Space
NARX-NN	TDL Buffer Size k	10	{1, 5, 10, 15}
	MLP Hidden Layers L	3	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
	MLP Neurons H_M	128	{32, 64, 128, 256}
LSTM-MLP	TDL Buffer Size k	10 or 0	{1, 5, 10, 15}
	MLP Hidden Layers L	2	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
	MLP Neurons H_M	128	{32, 64, 128, 256}
MA-LSTM-MLP	LSTM Cells H_L	64	{32, 64, 128}
	TDL Buffer Size k	10	{1, 5, 10, 15}
	MLP Hidden Layers L	2	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
MA-LSTM-MLP	MLP Neurons H_M	128	{32, 64, 128, 256}
	LSTM Cells H_L	64	{32, 64, 128}
	Heads h	4	{2, 4, 8}
	Key Vector Dimension d_k	8	{4, 8, 16}
Training	Epochs e	200	{100, 200, 300}
	Batch Size B	16	{8, 16, 32}
	Learning Rate η	1e-4	{1e-5, 1e-4, 1e-3}
	Cost Function J	MSE	{MSE, R^2 }
	Early Stop s	3	{3, 5, 10}

TABLE IV: RMSE obtained in the Wiener-Hammerstein Benchmark with different architectures. The proposed MA-LSTM-MLP architecture outperforms the alternatives. The value in bold shows the best result.

Architecture	RMSE
NARX-NN, $k = 10$ (Baseline)	0.0633
LSTM-MLP, $k = 0$	0.0254
LSTM-MLP, $k = 10$	0.0225
MA-LSTM-MLP, $k = 10$	0.0208
STORN [57]	0.0425
STORN with BiGRU [58]	0.023

TDL and adding a Multi-Head Attention layer. The specific hyperparameter values for the MA-LSTM-MLP and for the other architectures are summarized in Table III.

Table IV shows the RMSE obtained in the Wiener-Hammerstein benchmark with the baseline, the proposed architecture and state-of-the-art NN-based solutions. The reason for using RMSE is that it is the recommended metric of this benchmark to measure performance. The RMSE shown is the error obtained by predicting the 16,384 time instants of the two test scenarios and averaging the result. In our experiments, the predictions are made knowing only the state of the system at the initial time instant y_0 and iterating the architecture for each of the external inputs $U_{0,T-1}$ for the whole prediction horizon T , which is 16,384 in this case. As can be seen, the proposed MA-LSTM-MLP architecture obtains lower error compared to the rest of the alternatives. The LSTM-MLP architecture also achieves a good result with and without TDL, outperforming the baseline NARX-NN architecture.

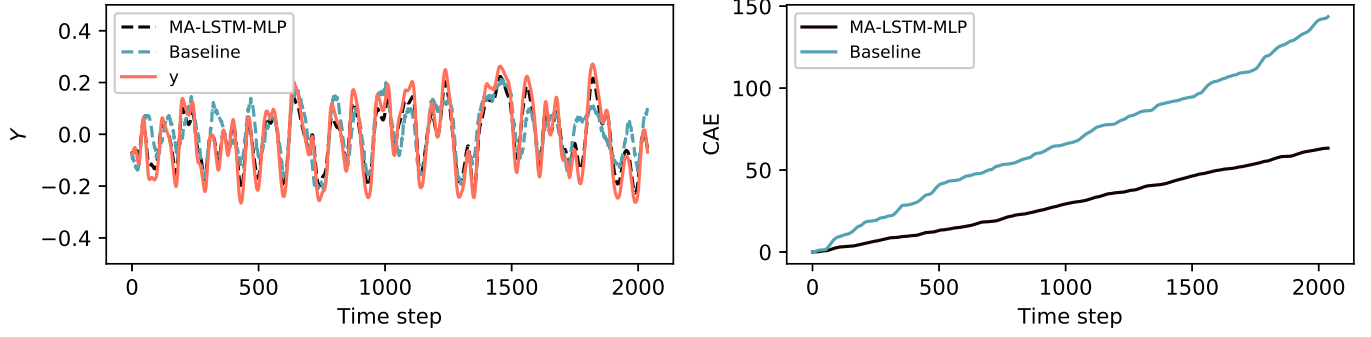
The Wiener-Hammerstein Process Noise System benchmark has been widely used in the literature as a test for the simulation of nonlinear systems. In black box modeling, the work of [57] compares different NN architectures applied in this benchmark, obtaining the best results using the Stochastic RNN (STORN) architecture [61]. STORN obtained an RMSE of 0.0425, being outperformed by the MA-LSTM-MLP archi-

ture proposed here.

In [58] it is proposed to use Bidirectional Gated Recurrent Unit (BiGRU) in the STORN architecture, instead of using basic RNN. The main advantage of Bidirectional RNN is that they not only use current and past external inputs in each iteration, but also analyze future inputs. The problem with using future inputs is that we would not meet the requirements described in our paper: The architecture could not be used to monitor the system in real time, and would not be able to adapt to unexpected external changes. Using STORN with BiGRU is a state-of-the-art black box modeling solution in the Wiener-Hammerstein Process Noise System benchmark, with an RMSE of 0.023. Our MA-LSTM-MLP architecture reduces this error to 0.0208 and do not use future inputs for the estimation of each time step.

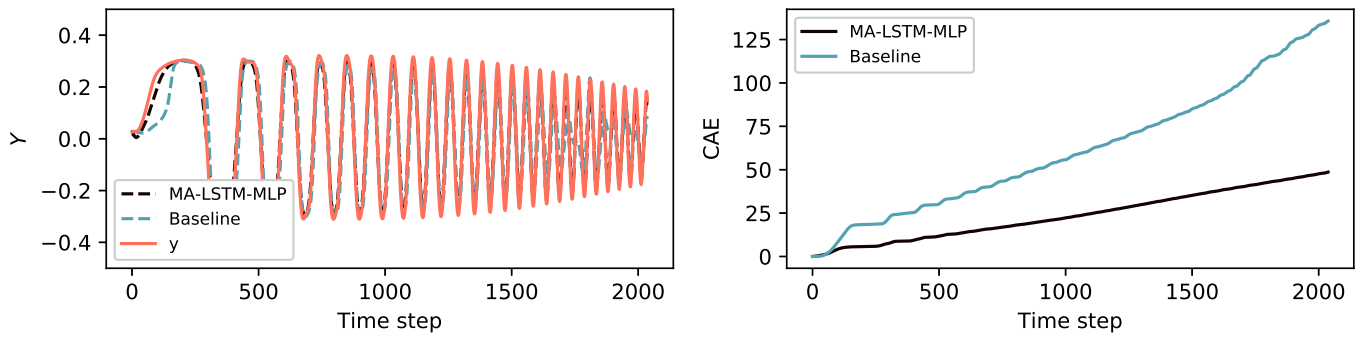
A comparison between the MA-LSTM-MLP and baseline predictions with the actual system state in the first 2048 time steps of the two benchmark tests is shown in Fig. 5. This figure also shows the evolution of the CAE of the baseline and the MA-LSTM-MLP architecture over the prediction horizon for both benchmark scenarios. The MA-LSTM-MLP architecture produces a negligible error compared to the real system states in both benchmark scenarios, outperforming the baseline architecture in both cases. The baseline architecture performs a prediction with noise and errors in the Multisine benchmark case. The Swept sine benchmark scenario is simpler, and the baseline model also performs correctly during most of the simulation, however, this architecture shows perceptible errors in the first 200 time steps and from time step 1,700 onwards, while the MA-LSTM-MLP maintains a negligible error during the entire system simulation. Observing the evolution of the CAE during all time steps, it is confirmed that our proposed architecture is robust to long prediction horizons. It can be seen in both scenarios how the CAE in the MA-LSTM-MLP case grows approximately linearly and stably over the time steps, indicating that the absolute error at each time step is low and approximately constant. In the baseline architecture,

Wiener-Hammerstein Multisine Benchmark



(a) Prediction and CAE of the first 2048 time steps of the Wiener-Hammerstein Multisine Benchmark with the baseline architecture and the proposed MA-LSTM-MLP.

Wiener-Hammerstein Swept sine Benchmark



(b) Prediction and CAE of the first 2048 time steps of the Wiener-Hammerstein Swept sine Benchmark with the baseline architecture and the proposed MA-LSTM-MLP.

Fig. 5: Prediction and CAE of the proposed MA-LSTM-MLP architecture and the baseline for the first 2048 time steps of the Multisine (5a) and the Swept sine (5b) Benchmark. The MA-LSTM-MLP architecture produces a negligible error compared to the true state of the system, outperforming the baseline which has higher noise and error prediction.

the CAE is higher throughout the prediction horizon and is also unstable, growing more as the prediction horizon increases.

C. Industrial Robot

The Industrial Robot benchmark [62] consists of data collected from a KUKA KR300 R2500 ultra SE industrial robot, which has a nominal payload capacity of 300 kg and a reach of 2500 mm. The dataset has 43,622 time steps with a sampling rate of 10 Hz. At each time instant, we have the information of six motor torques τ [Nm] and six joint positions q [deg]. The experimental design incorporates 36 different robot trajectories, executed twice, to ensure repeatability. These trajectories were optimized under physical constraints such as position, velocity, acceleration, and jerk limits. Data post-processing includes filtering and resampling to 100 ms intervals. The simulation problem consists in estimating at each time instant the joint position from the motor torques.

This benchmark consists of a problem with highly nonlinear dynamics, and it is a complex system with multi-input and multi-output. Another major difficulty of the benchmark is that it is easy to over-fit the solutions to the training data.

The test data consists of 3,635 time steps. The remaining 39,987 time steps have been divided, using 36,387 for training

and 3,600 for validation. During the training phase, in order to be able to parallelize the training, a prediction horizon of 600 has been set, dividing all time steps into series of this length.

To make the comparisons between the proposed architecture and the baseline, the same hyperparameters related to the training have been used and the number of parameters of the proposed network are in the same order of magnitude (Table V). Adam has been chosen as the optimizer.

The strategy for choosing the hyperparameters related to the architecture was similar to the previous example. First, a gridsearch was performed to fit the NARX-NN architecture and then those parameters were kept in the MLP layers and the rest of the hyperparameters were optimized for the LSTM-MLP and MA-LSTM-MLP architectures. The final hyperparameters values for the architectures for the Industrial Robot Benchmark are summarized in Table V.

Table VI shows the NRMSE obtained with the architectures developed in the Industrial Robot benchmark test data. NRMSE has been used since it is the recommended metric in this benchmark. The NRMSE shown is the error obtained by predicting the 3,635 time steps of the test data. As in the previous example, all prediction is performed knowing only the state of the robot at the initial instant y_0 and iterating the

TABLE V: Hyperparameters of the three architectures and their training configurations for the Industrial Robot Benchmark, including the specific values explored during the grid search.

Architecture	Hyperparameter	Value	Search Space
NARX-NN	TDL Buffer Size k	10	{1, 5, 10, 15}
	MLP Hidden Layers L	3	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
	MLP Neurons H_M	512	{128, 256, 512}
LSTM-MLP	TDL Buffer Size k	10 or 0	{1, 5, 10, 15}
	MLP Hidden Layers L	2	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
	MLP Neurons H_M	512	{128, 256, 512}
MA-LSTM-MLP	LSTM Cells H_L	128	{64, 128, 256}
	TDL Buffer Size k	10	{1, 5, 10, 15}
	MLP Hidden Layers L	2	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
Training	MLP Neurons H_M	512	{128, 256, 512}
	LSTM Cells H_L	128	{64, 128, 256}
	Heads h	8	{4, 8}
	Key Vector Dimension d_k	24	{8, 16, 24, 32}
Training	Epochs e	200	{100, 200, 300}
	Batch Size B	6	{2, 3, 6}
	Learning Rate η	1e-4	{1e-5, 1e-4, 1e-3}
	Cost Function J	R ²	{MSE, R ² }
Training	Early Stop s	5	{3, 5, 10}

TABLE VI: NRMSE obtained in the Industrial Robot Benchmark with different architectures. The proposed MA-LSTM-MLP architecture obtains the best results. The value in bold shows the best result.

Architecture	NRMSE
NARX-NN, $k = 10$ (Baseline)	0.146
LSTM-MLP, $k = 0$	0.158
LSTM-MLP, $k = 10$	0.142
MA-LSTM-MLP, $k = 10$	0.125

architecture for each of the external inputs of $U_{0,T-1}$. We can observe how the architecture that obtains more error is the LSTM-MLP with $k = 0$, indicating that in this dataset it is relevant to have a TDL buffer with the last estimated states of the system. The LSTM-MLP architecture with $k = 10$ slightly reduces the error obtained by the NARX-NN architecture with the same TDL size. As in the previous case, exploiting the potential of using TDL buffers, the proposed MA-LSTM-MLP architecture obtains the least NRMSE.

In Fig. 6 the prediction of the MA-LSTM-MLP and baseline architectures for 600 random time steps in the test data is shown, compared to the true six joint position. The proposed architecture has a small error with respect to the real states of the system, while the baseline solution has more error and noise in its predictions. The CAE has been calculated individually for each of the six joint positions, showing that the proposed model is more robust and obtains less error over the entire prediction horizon. In the baseline case, the CAE is higher along all time steps, and the difference increases as this prediction horizon increases.

With this dataset, in the context of multistep prediction with NN, the previous work of [63] has to be highlighted. This work proposes Recurrent Linear Parameter-Varying Networks (ReLiNet), an RNN with faithful explanations and stability guarantees. In addition, in that paper, they also use LSTM as baseline to compare their proposal. The multistep prediction tests are performed in that paper with the following conditions:

The previous 60 time steps are known, both the system state and the external inputs, and the next system states are predicted from the next 60 external inputs. With this context in the Industrial Robot benchmark, ReLiNet and the LSTM architecture trained in that paper obtain an NRMSE of 0.548 and 0.409 respectively at the prediction horizon of 60. As we have seen, with our approach, we outperform these architectures by reducing this error to 0.125 in the case of the proposed MA-LSTM-MLP and starting from only one known time step and predicting the 3,635 time steps of the test data. As we can observe, the proposed architecture without Multi-Head Attention (LSTM-MLP), both with and without TDL buffer, as well as the NARX-NN baseline architecture, also obtain less error, reinforcing our proposed iterative approach using the no-TF training algorithm.

D. Dryer

The Dryer dataset consists of a set of data obtained from real-world experiments. The description of the experiments as well as the data are publicly available for download in the DaISy repository on the website of Katholieke Universiteit Leuven ¹. The dryer dataset consists of a system with a single input and a single output, obtained from a laboratory setup acting as a hair dryer. The input is the voltage at the heating device, which consists of a mesh of resistance wires. The output is the air temperature, measured with a thermocouple.

In the previous benchmarks, we had a large number of data and time steps, which is very beneficial for NN training, whereas in this dataset, we have only 1000 time steps. This is an interesting situation to study if the proposed architecture is able to correctly model the system with limited data. We used the first 500 time steps to train and validate the network, and the last 500 for the test phase. During training, a prediction horizon of 100 time steps has been set, where the first 400

¹<https://homes.esat.kuleuven.be/~smc/daisy/index.html>

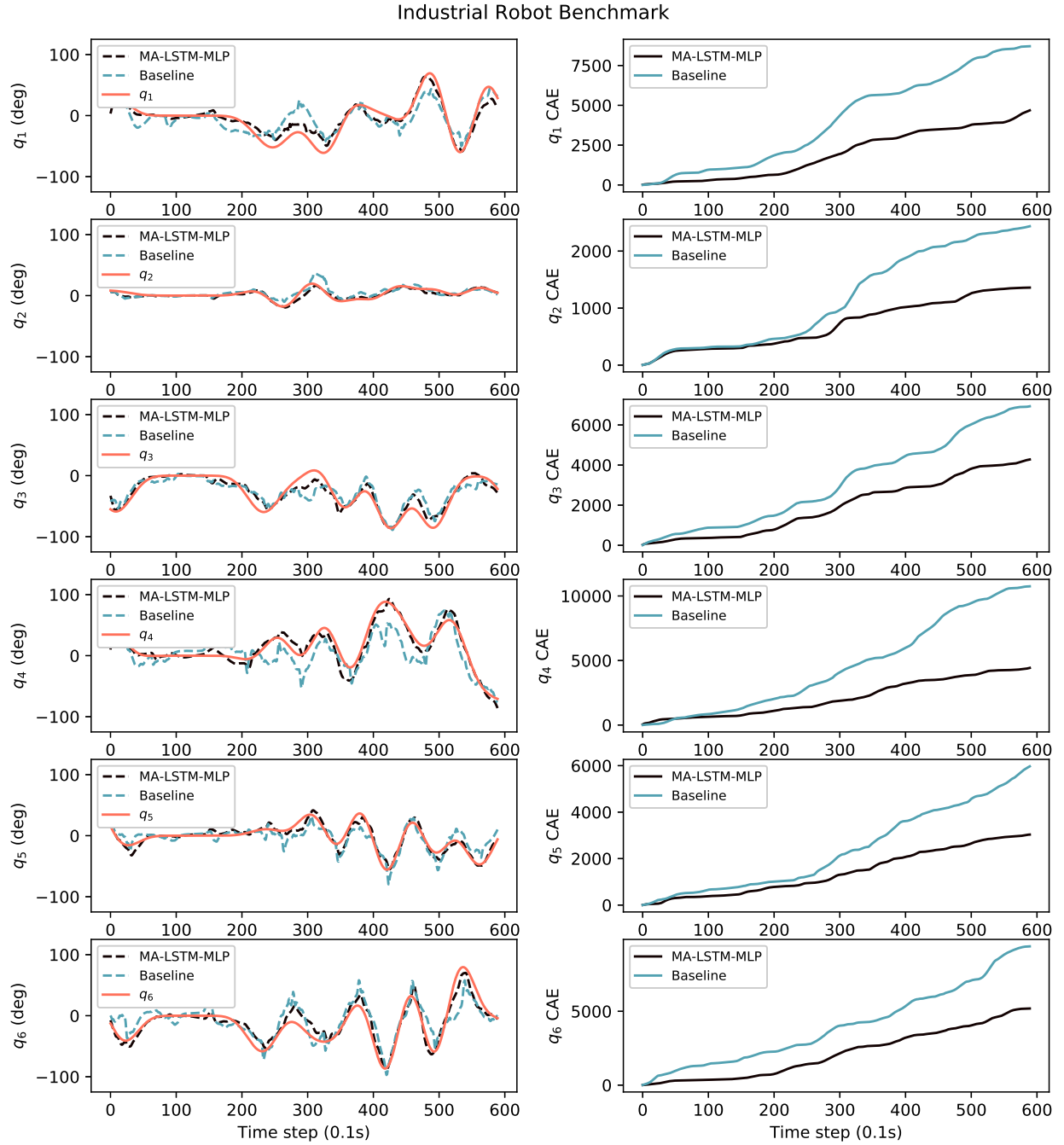


Fig. 6: Prediction and CAE of the MA-LSTM-MLP and baseline architecture for a random 600 time steps of the Industrial Robot Benchmark test data. The MA-LSTM-MLP architecture outperforms the baseline with less error and more stability.

time steps have been used for training and the next 100 for validation. During the test, the 500 time steps are predicted.

Again, to make fair comparisons between the proposed architecture and the baseline, we have used the same hyperparameters related to training and the networks have the same order of magnitude of parameters. To find the architecture-related hyperparameters, a gridsearch was performed to fit the baseline NARX-NN architecture. The hyperparameters found were set for the MLP layers and the remaining hyperparameters were optimized for the LSTM-MLP and MA-LSTM-MLP

architectures. The final hyperparameters, both those related to training and architectures, together with the search spaces can be found in Table VII. Adam has been used as optimizer.

Table VIII shows the RMSE obtained with the architectures trained on the Dryer dataset. The RMSE shown is the error obtained by predicting the 500 time steps of the test data. The prediction is performed knowing only the state of the dryer at the initial instant of the test data y_0 and iterating the architecture for each of the known external parameters $U_{0,T-1}$. We observe a trend similar to the previous examples.

TABLE VII: Hyperparameters of the three architectures and their training configurations for the Dryer Dataset, including the specific values explored during the grid search.

Architecture	Hyperparameter	Value	Search Space
NARX-NN	TDL Buffer Size k	10	{1, 5, 10, 15}
	MLP Hidden Layers L	3	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
	MLP Neurons H_M	256	{64, 128, 256, 512}
LSTM-MLP	TDL Buffer Size k	10 or 0	{1, 5, 10, 15}
	MLP Hidden Layers L	2	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
	MLP Neurons H_M	256	{64, 128, 256, 512}
MA-LSTM-MLP	LSTM Cells H_L	64	{32, 64, 128, 256}
	TDL Buffer Size k	10 or 0	{1, 5, 10, 15}
	MLP Hidden Layers L	2	{1, 2, 3, 4}
	MLP Activation σ	ReLU	{ReLU, Tanh, Sigmoid}
Training	MLP Neurons H_M	256	{64, 128, 256, 512}
	LSTM Cells H_L	64	{32, 64, 128, 256}
	Heads h	4	{2, 4, 8}
	Key Vector Dimension d_k	8	{4, 8, 16, 32}
Training	Epochs e	200	{100, 200, 300}
	Batch Size B	1	{1, 2}
	Learning Rate η	1e-4	{1e-5, 1e-4, 1e-3}
	Cost Function J	MSE	{MSE, R^2 }
	Early Stop s	5	{3, 5, 10}

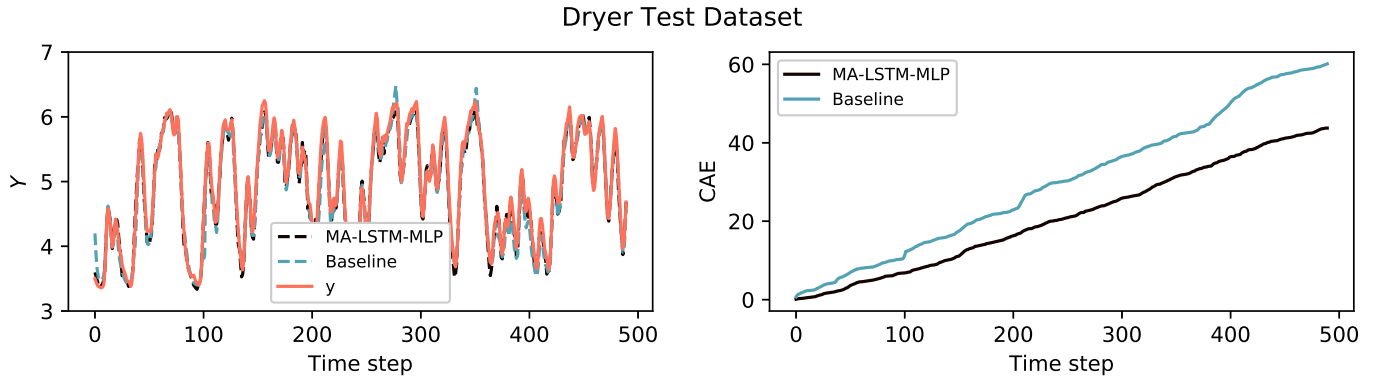


Fig. 7: Prediction and CAE of the MA-LSTM-MLP and baseline architecture for the 500 time steps of the Dryer Test Dataset. The MA-LSTM-MLP architecture outperforms the baseline with less error and more stability.

TABLE VIII: RMSE obtained in the Dryer Test Dataset with different architectures. The proposed MA-LSTM-MLP architecture outperforms the alternatives. The value in bold shows the best result.

Architecture	RMSE
NARX-NN, $k = 10$ (Baseline)	0.1392
LSTM-MLP, $k = 0$	0.1827
LSTM-MLP, $k = 10$	0.1164
MA-LSTM-MLP, $k = 10$	0.1102
RFS-LSSVR [29]	0.1130

The highest error is obtained using LSTM-MLP with $k = 0$, showing that in this example it is also relevant to keep the TDL buffers with the last estimated states of the system. The rest of the architectures present the expected behavior. LSTM-MLP with TDL buffers decreases the error obtained by the baseline. The proposed MA-LSTM-MLP model again obtains less error than the other alternatives.

In the state of the art a variation of a squares support vector regression model is proposed for the identification and modeling of dynamical systems [29], RFS-LSSVR. The main

feature of their proposal is the tolerance to outliers in the input of the system. In the Dryer test dataset, RFS-LSSVR without outliers obtains an RMSE of 0.1130, quite similar but slightly higher than MA-LSTM-MLP. Being a small dataset, it is difficult to lower the error further. Our proposal proves to work well in these scenarios with limited data.

Fig. 7 shows the prediction of the MA-LSTM-MLP and baseline architectures for the 500 time steps of the Dryer test dataset, compared to the true system state Y . We can see that both architectures model the system correctly. However, in the baseline architecture we observe more noise and outliers that disappear in the MA-LSTM-MLP prediction, where the error is negligible. The CAE plot confirms these observations, where we see that the proposed architecture is more robust and obtains less error during the whole prediction horizon. In this plot the baseline has higher CAE during the whole prediction horizon and the difference increases with respect to the time steps.

TABLE IX: Computational cost of the architectures studied in the benchmarks. The number of parameters corresponds to the weights and bias. Execution time corresponds to 100 iterations of the architectures using an NVIDIA Titan XP. Although MA-LSTM-MLP has a more complex design, it is efficient in memory usage and computational cost.

Benchmark	Architecture	Parameters	Time (ms)
Wiener-Hammerstein	NARX-NN, $k = 10$	35,841	302.29
	LSTM-MLP, $k = 0$	42,113	342.32
	LSTM-MLP, $k = 10$	46,721	356.32
	MA-LSTM-MLP, $k = 10$	52,195	551.21
Industrial Robot	NARX-NN, $k = 10$	559,622	329.45
	LSTM-MLP, $k = 0$	403,974	331.42
	LSTM-MLP, $k = 10$	459,270	357.96
	MA-LSTM-MLP, $k = 10$	530,514	566.04
Dryer	NARX-NN, $k = 10$	137,217	306.28
	LSTM-MLP, $k = 0$	99,841	374.61
	LSTM-MLP, $k = 10$	104,449	390.93
	MA-LSTM-MLP, $k = 10$	109,923	564.51

TABLE X: Mean Absolute Error (MAE), Median of the Absolute Error (Med AE) and Standard Deviaton of the Absolute Error (STD AE) of the proposed architectures in the three datasets studied. The metrics are obtained from the prediction of the architectures of all time steps of the test data. In all cases, we observe how MA-LSTM-MLP outperforms the alternatives. The values in bold show the best results.

Benchmark	Architecture	MAE	Med AE	STD AE
Wiener-Hammerstein	NARX-NN, $k = 10$	0.052	0.036	0.047
	LSTM-MLP, $k = 0$	0.021	0.013	0.022
	LSTM-MLP, $k = 10$	0.019	0.012	0.018
	MA-LSTM-MLP, $k = 10$	0.017	0.008	0.017
Industrial Robot	NARX-NN, $k = 10$	14.90	8.97	17.65
	LSTM-MLP, $k = 0$	13.98	8.38	16.23
	LSTM-MLP, $k = 10$	12.33	7.21	14.65
	MA-LSTM-MLP, $k = 10$	10.83	6.46	12.58
Dryer	NARX-NN, $k = 10$	0.125	0.111	0.089
	LSTM-MLP, $k = 0$	0.144	0.115	0.111
	LSTM-MLP, $k = 10$	0.109	0.104	0.071
	MA-LSTM-MLP, $k = 10$	0.089	0.077	0.064

E. Computational Cost

We measured the time for the proposed architecture to perform 100 iterations to estimate the computational cost in the benchmarks. Table IX shows the execution time in milliseconds for 100 iterations of all studied architectures. Note that an iteration computes in parallel batch size B time steps, one for each series. In addition, the table shows the number of trainable parameters (weights and bias) that each architecture has, in order to estimate the memory cost. The architectures have the hyperparameters detailed in the sections of each benchmark (Section IV-B, Section IV-C and Section IV-D). Although MA-LSTM-MLP has a more complex design, overall it is efficient in memory usage and computational cost. All the architectures shown are efficient in computational cost, since if we analyze the worst case (MA-LSTM-MLP on the Industrial Robot Benchmark) we obtain that we predict 100 time steps in 566 ms. This is 5.66 ms per time step, or 176.67 Hz. All computational cost tests were performed using an NVIDIA Titan XP graphics card.

F. Statistical Analysis

In the sections related to the results of the architectures in the different datasets (Section IV-B, Section IV-C and Section IV-D) only one metric is shown for each of the architectures compared. The purpose of showing only one metric in the results tables is to make comparisons in a simple way, using the metric recommended by each of the datasets. In

this subsection we perform a more detailed statistical analysis of the results obtained in each of the datasets with the different architectures studied.

Table X shows Mean Absolute Error (MAE), Median of the Absolute Error (Med AE) and Standard Deviation of the Absolute Error (STD AE) of the proposed architectures in the three datasets studied. The metrics are calculated from the prediction of all time steps of the test datasets. We can observe how the different parts of the proposed architecture improve the results. The case of the baseline architecture (NARX-NN), in all the benchmarks is improved by the hybrid alternative in which we add LSTM layers, especially with LSTM-MLP where $k = 10$. In addition, in all datasets we obtain the best result by adding the attention layer, with our proposal MA-LSTM-MLP.

In addition to the table with metrics related to the absolute error, we present for each of the analyzed datasets a boxplot with the error obtained by each architecture. Each architecture predicts all the time steps of the test data. With the boxplots, we can observe and compare the dispersion and range of the error, as well as whether the error is symmetrical or has a bias. It is also important to consider the Interquartile Range (IQR). The length of the box shows the IQR, which represents the middle 50% of the errors. A smaller IQR indicates less variability in the errors, which is desirable.

Fig. 8 shows the boxplot of the error in the Wiener-Hammerstein Benchmark. As can be seen, all alternatives

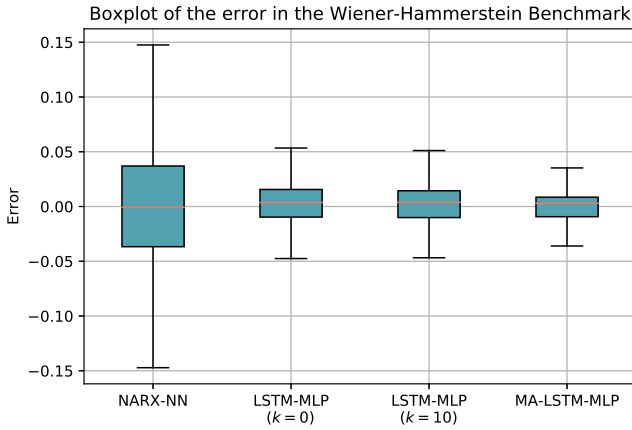


Fig. 8: Boxplot of the error in the Wiener-Hammersrtein Benchmark. All the architectures show a symmetric error, with the median close to 0. It is observed that MA-LSTM-MLP obtains a smaller IQR and smaller range of maximum and minimum error values, demonstrating higher accuracy.

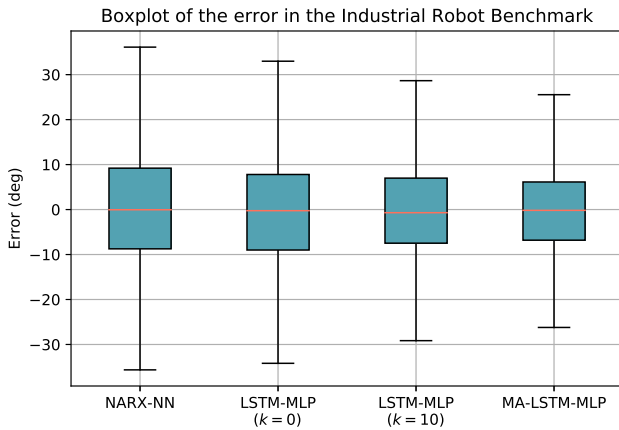


Fig. 9: Boxplot of the error in the Industrial Robot Benchmark. All the architectures show a symmetric error, with the median close to 0. MA-LSTM-MLP obtains a smaller IQR and smaller range of maximum and minimum error values, demonstrating higher accuracy.

show a median close to 0, with a symmetrical error. Regarding the IQR and the range of maximum and minimum values, we observe the best results in the case of MA-LSTM-MLP, obtaining smaller values than the alternatives. We can see similar conclusions in Fig. 9, in the case of the Industrial Robot Benchmark.

Fig. 10 shows the boxplot of the error in the Dryer test dataset. In this case, the four alternatives show an error with a positive bias. MA-LSTM-MLP is the best option, showing a lower IQR and range of errors than the alternatives, and above all with a median closer to 0.

G. Discussion

The results presented in the previous sections demonstrate the effectiveness of the proposed RNN-based architecture in simulating complex nonlinear dynamical systems. As initially

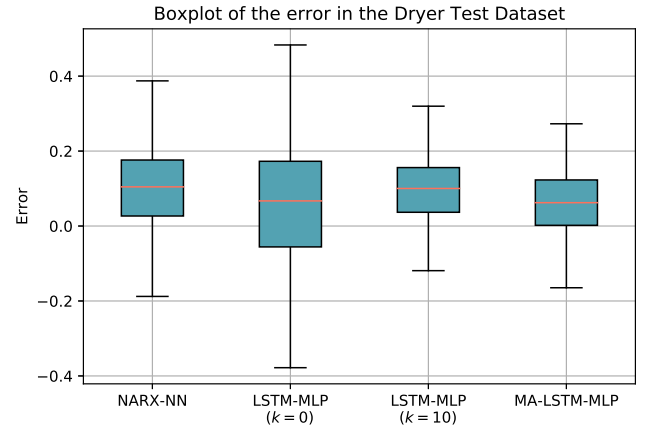


Fig. 10: Boxplot of the error in the Dryer test dataset. All architectures show a positively biased error. It is observed that MA-LSTM-MLP obtains a lower IQR, with values closer to 0 than the alternatives, showing higher accuracy.

defined in the paper, the main requirement for the architectures is that they should predict the behavior of a dynamical system knowing only the initial state and any number of time steps of external inputs. We have shown an architecture with an iterative approach is well suited for this problem.

The first architecture shown is LSTM-MLP. This network is similar to the baseline NARX-NN, to which we have added an LSTM layer. In our experiments, this network shows significant improvements with respect to the NARX-NN architecture, obtaining less error and reducing the number of neurons and parameters. This architecture is part of our final proposal, MA-LSTM-MLP.

The proposed MA-LSTM-MLP architecture consistently outperforms LSTM-MLP and NARX-NN on the Wiener-Hammerstein Process Noise System [59], Industrial Robot [62], and Dryer benchmarks. We propose this architecture with the idea of exploiting information from TDL buffers that store the latest external inputs and estimated system states. The results highlight the effectiveness of integrating attention mechanisms to capture complex temporal dependencies and improve the modeling of dynamic systems.

The proposed architecture is computationally efficient. In particular, with the iterative approach, the function of the architectures at each iteration is to predict the state at the next time step and not to compute all at once a large prediction window as it would be in the case of a sequence-to-sequence problem. All experiments shown in the paper are with a TDL buffer size of 10, limiting the size of the input and output data. The LSTM-MLP architecture reduces the number of parameters with respect to the NARX-NN baseline, obtaining better performance. Additionally, the Multi-Head Attention layer does not add many parameters since the TDLs are limited to size 10.

The real computational cost of these architectures is in the training, since training with no-TF is computationally more demanding than training with TF. However, training with no-TF provide architectures that work well with the iterative approach. We obtain architectures that practically

do not lose accuracy over large prediction horizons (16,384 time steps in the case of the Wiener-Hammerstein Benchmark and 3,635 in the case of the Industrial Robot Benchmark). Training with TF optimizes the architectures for single time step predictions, causing the error to increase greatly as we increase the prediction horizon.

The generalization capacity of our architectures is shown in their high performance in the tests, adapting to situations that have not been observed during the training phase. This ability of the architectures to generalize across different datasets and adapt to unseen variations in the system dynamics is necessary for real-world deployment.

V. CONCLUSION

In this paper, we have proposed an RNN architecture that simulates complex nonlinear dynamical systems. Our goal is to predict the behavior of the system knowing only the initial state of the system and any number of time steps of external inputs, being this number the prediction horizon of the system. State-of-the-art solutions using RNN are limited by fixed prediction horizons and the lack of novel approaches.

We propose the MA-LSTM-MLP architecture. This architecture involves iterative state estimation, where in each iteration only the next time step is calculated. This approach works for large prediction horizons by training the architecture with a no-TF approach.

Our hybrid MA-LSTM-MLP architecture combines LSTM layers with MLP to capture long-term dependencies and nonlinearities of the dynamical system. In addition, including a Multi-head Attention layer allows exploiting information from previous estimated states and external inputs, improving the accuracy and stability. Our proposed architecture outperforms the baseline NARX-NN architecture as well as different state-of-the-art solutions in three public benchmarks.

Our study demonstrates the potential of using the MA-LSTM-MLP architecture in simulating nonlinear dynamical systems. In view of the good performance of the proposed architecture on datasets with real-world data, we consider as future work to implement this architecture in real-world applications, thus addressing modeling, identification and control of systems in practical environments.

ACKNOWLEDGEMENTS

This work was funded by the Government of Aragón, group T45_23R, and by projects CPP2021-008938, PID2021-124137OB-I00 and TED2021-130224B-I00, funded by MCIN/AEI/10.13039/501100011033 and by the European Union-NextGenerationEU/PRTR.

AUTHOR CONTRIBUTIONS

Javier Fañanás-Anaya: Conceptualization, Methodology, Software, Validation, Writing - Original Draft. Gonzalo López-Nicolás: Conceptualization, Writing - Review & Editing, Supervision, Project Administration, Funding Acquisition. Carlos Sagüés: Conceptualization, Writing - Review & Editing, Supervision, Project Administration, Funding Acquisition.

DATA AVAILABILITY STATEMENT

The code of the proposed architecture, pre-trained models, datasets, and a tutorial for adding new datasets are publicly available at <https://github.com/javierfa98/MA-LSTM-MLP>.

FINANCIAL INTEREST

The authors declare they have no financial interests.

REFERENCES

- [1] C. Mavrogiannis, F. Baldini, A. Wang, D. Zhao, P. Trautman, A. Steinfeld, and J. Oh, "Core Challenges of Social Robot Navigation: A Survey," *ACM Transactions on Human-Robot Interaction*, vol. 12, no. 3, pp. 36:1–36:39, 2023.
- [2] Y. Wu, W. Niu, L. Kong, X. Yu, and W. He, "Fixed-time neural network control of a robotic manipulator with input deadzone," *ISA Transactions*, vol. 135, pp. 449–461, 2023.
- [3] A. G. Olabi, A. A. Abdelghafar, H. M. Maghrabie, E. T. Sayed, H. Rezk, M. A. Radi, K. Obaideen, and M. A. Abdelkareem, "Application of artificial intelligence for prediction, optimization, and control of thermal energy storage systems," *Thermal Science and Engineering Progress*, vol. 39, p. 101730, 2023.
- [4] E. A. Antonelo, E. Camponogara, L. O. Seman, J. P. Jordanou, E. R. de Souza, and J. F. Hübner, "Physics-informed neural nets for control of dynamical systems," *Neurocomputing*, vol. 579, p. 127419, 2024.
- [5] T. O. Kehinde, F. T. S. Chan, and S. H. Chung, "Scientometric review and analysis of recent approaches to stock market forecasting: Two decades survey," *Expert Systems with Applications*, vol. 213, p. 119299, 2023.
- [6] A. T. Oyewole, O. B. Adeoye, W. A. Addy, C. C. Okoye, O. C. Ofodile, and C. E. Ugochukwu, "Predicting stock market movements using Neural Networks: A review and application study," *Computer Science & IT Research Journal*, vol. 5, no. 3, pp. 651–670, 2024.
- [7] K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian, "Accurate medium-range global weather forecasting with 3D neural networks," *Nature*, vol. 619, no. 7970, pp. 533–538, Jul. 2023.
- [8] L. Chen, X. Zhong, F. Zhang, Y. Cheng, Y. Xu, Y. Qi, and H. Li, "FuXi: a cascade machine learning forecasting system for 15-day global weather forecast," *npj Climate and Atmospheric Science*, vol. 6, no. 1, pp. 1–11, 2023.
- [9] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Systems with Applications*, vol. 207, p. 117921, 2022.
- [10] K.-H. N. Bui, J. Cho, and H. Yi, "Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues," *Applied Intelligence*, vol. 52, no. 3, pp. 2763–2774, 2022.
- [11] S. K. Nayak, A. Bit, A. Dey, B. Mohapatra, and K. Pal, "A Review on the Nonlinear Dynamical System Analysis of Electrocardiogram Signal," *Journal of Healthcare Engineering*, vol. 2018, no. 1, p. 6920420, 2018.
- [12] P. M. Datilo, Z. Ismail, and J. Dare, "A Review of Epidemic Forecasting Using Artificial Neural Networks," *Epidemiology and Health System Journal*, vol. 6, no. 3, pp. 132–143, 2019.
- [13] C. Legaard, T. Schranz, G. Schweiger, J. Drgoña, B. Falay, C. Gomes, A. Iosifidis, M. Abkar, and P. Larsen, "Constructing Neural Network Based Models for Simulating Dynamical Systems," *ACM Computing Surveys*, vol. 55, no. 11, pp. 236:1–236:34, 2023.
- [14] I. Hammoud, S. Hentzelt, T. Oehlschlaegel, and R. Kennel, "Learning-based model predictive current control for synchronous machines: An LSTM approach," *European Journal of Control*, vol. 68, p. 100663, 2022.
- [15] T. V. Baby, S. M. Sotoudeh, and B. HomChaudhuri, "Data-Driven Prediction and Predictive Control Methods for Eco-Driving in Production Vehicles," *IFAC-PapersOnLine*, vol. 55, no. 37, pp. 633–638, 2022.
- [16] S. Ye, C. Wang, Y. Wang, X. Lei, X. Wang, and G. Yang, "Real-time model predictive control study of run-of-river hydropower plants with data-driven and physics-based coupled model," *Journal of Hydrology*, vol. 617, p. 128942, 2023.
- [17] I. Lenz, R. Knepper, and A. Saxena, "DeepMPC: Learning Deep Latent Features for Model Predictive Control," in *Robotics: Science and Systems XI*, Rome, Italy, 2015.
- [18] J. Wu, H. Peng, Q. Chen, and X. Peng, "Modeling and control approach to a distinctive quadrotor helicopter," *ISA Transactions*, vol. 53, no. 1, pp. 173–185, 2014.

- [19] Y. Kang, S. Chen, X. Wang, and Y. Cao, "Deep Convolutional Identifier for Dynamic Modeling and Adaptive Control of Unmanned Helicopter," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, pp. 524–538, 2019.
- [20] N. Mohajerin and S. L. Waslander, "Multistep Prediction of Dynamic Systems With Recurrent Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3370–3383, 2019.
- [21] S. Looper and S. L. Waslander, "Temporal Convolutions for Multi-Step Quadrotor Motion Prediction," in *2022 19th Conference on Robots and Vision (CRV)*. Toronto, ON, Canada: IEEE, 2022, pp. 32–39.
- [22] Y. Zhao, C. Jiang, M. A. Vega, M. D. Todd, and Z. Hu, "Surrogate Modeling of Nonlinear Dynamic Systems: A Comparative Study," *Journal of Computing and Information Science in Engineering*, vol. 23, no. 1, p. 011001, 2023.
- [23] R. Shobana, R. Kumar, and B. Jaint, "A recurrent neural network-based identification of complex nonlinear dynamical systems: a novel structure, stability analysis and a comparative study," *Soft Computing*, 2023.
- [24] X. Huang, Q. Li, Y. Tai, Z. Chen, J. Zhang, J. Shi, B. Gao, and W. Liu, "Hybrid deep neural model for hourly solar irradiance forecasting," *Renewable Energy*, vol. 171, pp. 1041–1060, 2021.
- [25] Y. Ding, Y. Zhu, J. Feng, P. Zhang, and Z. Cheng, "Interpretable spatio-temporal attention LSTM model for flood forecasting," *Neurocomputing*, vol. 403, pp. 348–359, 2020.
- [26] Y. Li, Z. Zhu, D. Kong, H. Han, and Y. Zhao, "EA-LSTM: Evolutionary attention-based LSTM for time series prediction," *Knowledge-Based Systems*, vol. 181, p. 104785, 2019.
- [27] G. Chen, W. Peng, and F. Yang, "An LSTM-SA model for SOC estimation of lithium-ion batteries under various temperatures and aging levels," *Journal of Energy Storage*, vol. 84, p. 110906, 2024.
- [28] S. A. A. Shah, S. G. Niazi, S. Deng, H. M. Hamza Azam, K. Mian Muhammad Yasir, J. Kumar, Z. Xu, and M. Wu, "A novel positional encoded attention-based Long short-term memory network for state of charge estimation of lithium-ion battery," *Journal of Power Sources*, vol. 590, p. 233788, 2024.
- [29] J. Santos and G. Barreto, "Novel sparse LSSVR models in primal weight space for robust system identification with outliers," *Journal of Process Control*, vol. 67, 2017.
- [30] C. L. C. Mattos and G. A. Barreto, "A stochastic variational framework for Recurrent Gaussian Processes models," *Neural Networks*, vol. 112, pp. 54–72, 2019.
- [31] J. A. Bessa, G. A. Barreto, and A. R. Rocha-Neto, "An Outlier-Robust Growing Local Model Network for Recursive System Identification," *Neural Processing Letters*, vol. 55, no. 4, pp. 4257–4289, 2023.
- [32] E. Akagündüz and O. Cifdaloz, "Dynamical system parameter identification using deep recurrent cell networks," *Neural Computing and Applications*, vol. 33, no. 23, pp. 16745–16757, 2021.
- [33] Q. Kang, E. J. Chen, Z.-C. Li, H.-B. Luo, and Y. Liu, "Attention-based LSTM predictive model for the attitude and position of shield machine in tunneling," *Underground Space*, vol. 13, pp. 335–350, 2023.
- [34] G. Li, J. Na, D. P. Stoten, and X. Ren, "Adaptive neural network feedforward control for dynamically substructured systems," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 3, pp. 944–954, 2014.
- [35] G. Peng, C. Yang, W. He, and C. L. P. Chen, "Force sensorless admittance control with neural learning for robots with actuator saturation," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 4, pp. 3138–3148, 2020.
- [36] Q. Liu, D. Li, S. S. Ge, and Z. Ouyang, "Adaptive feedforward neural network control with an optimized hidden node distribution," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 1, pp. 71–82, 2021.
- [37] F. Jamsheed and S. J. Iqbal, "Simplified artificial neural network based online adaptive control scheme for nonlinear systems," *Neural Computing and Applications*, vol. 35, no. 1, pp. 663–679, 2023.
- [38] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3223–3230.
- [39] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans, "Nonlinear Systems Identification Using Deep Dynamic Neural Networks," 2016.
- [40] T. Lin, B. Horne, P. Tino, and C. Giles, "Learning long-term dependencies in narx recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1329–1338, 1996.
- [41] H. Siegelmann, B. Horne, and C. Giles, "Computational capabilities of recurrent narx neural networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 27, no. 2, pp. 208–215, 1997.
- [42] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.
- [43] M. Wei, M. Ye, J. B. Li, Q. Wang, and X. Xu, "State of Charge Estimation of Lithium-Ion Batteries Using LSTM and NARX Neural Networks," *IEEE Access*, vol. 8, pp. 189236–189245, 2020.
- [44] A. Buevich, A. Sergeev, A. Shichkin, and E. Baglaeva, "A two-step combined algorithm based on NARX neural network and the subsequent prediction of the residues improves prediction accuracy of the greenhouse gases concentrations," *Neural Computing and Applications*, vol. 33, no. 5, pp. 1547–1557, 2021.
- [45] Y. Zhao, C. Jiang, M. A. Vega, M. D. Todd, and Z. Hu, "Surrogate Modeling of Nonlinear Dynamic Systems: A Comparative Study," *Journal of Computing and Information Science in Engineering*, vol. 23, no. 011001, 2022.
- [46] A. Cheng and Y. M. Low, "Improved generalization of NARX neural networks for enhanced metamodeling of nonlinear dynamic systems under stochastic excitations," *Mechanical Systems and Signal Processing*, vol. 200, p. 110543, 2023.
- [47] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [49] M.-H. Guo, T.-X. Xu, J.-J. Liu, Z.-N. Liu, P.-T. Jiang, T.-J. Mu, S.-H. Zhang, R. R. Martin, M.-M. Cheng, and S.-M. Hu, "Attention mechanisms in computer vision: A survey," *Computational Visual Media*, vol. 8, no. 3, pp. 331–368, 2022.
- [50] J. Fan, K. Zhang, Y. Huang, Y. Zhu, and B. Chen, "Parallel spatio-temporal attention-based TCN for multivariate time series prediction," *Neural Computing and Applications*, vol. 35, no. 18, pp. 13109–13118, 2023.
- [51] B. Lim, S. O. Arık, N. Loeff, and T. Pfister, "Temporal Fusion Transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021.
- [52] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [53] M. Sangiorgio and F. Dercole, "Robustness of LSTM neural networks for multi-step forecasting of chaotic time series," *Chaos, Solitons & Fractals*, vol. 139, p. 110045, 2020.
- [54] M. Sangiorgio, F. Dercole, and G. Guariso, "Forecasting of noisy chaotic systems with deep neural networks," *Chaos, Solitons & Fractals*, vol. 153, p. 111570, 2021.
- [55] F. Chollet *et al.*, "Keras," <https://keras.io>, 2015.
- [56] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, ser. OSDI'16, 2016, pp. 265–283.
- [57] D. Gedon, N. Wahlström, T. B. Schön, and L. Ljung, "Deep State Space Models for Nonlinear System Identification," *IFAC-PapersOnLine*, vol. 54, no. 7, pp. 481–486, 2021.
- [58] X. Liu, X. Du, X. Yang, and C. Cai, "Improved Stochastic Recurrent Networks for Nonlinear State Space System Identification," in *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*, Singapore, Singapore, 2023.
- [59] M. Schoukens and J. P. Noël, "Three Benchmarks Addressing Open Challenges in Nonlinear System Identification," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 446–451, 2017.
- [60] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2017.
- [61] J. Bayer and C. Osendorfer, "Learning Stochastic Recurrent Networks," 2015.
- [62] J. Weigand, J. Götz, J. Ulmen, and M. Ruskowski, "Dataset and Baseline for an Industrial Robot Identification Benchmark," in *6th Edition of the Workshop on Nonlinear System Identification Benchmarks*, Leuven, Belgium, 2022.
- [63] A. Baier, D. Aspandi, and S. Staab, "ReLiNet: Stable and Explainable Multistep Prediction with Recurrent Linear Parameter Varying Networks," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, Macao, P.R.China, 2023.