Neural Networks in RSCAD: Enhancing MMC-based HVDC Simulation with Advanced Machine Learning

Masalmeh, Bara; Prasad, Rashmi; Nougain, Vaibhav ; Lekić, Aleksandra

DOI
10.1109/TIA.2025.3529804

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Industry Applications

Citation (APA)
Masalmeh, B., Prasad, R., Nougain, V., & Lekić, A. (2025). Neural Networks in RSCAD: Enhancing MMC-based HVDC Simulation with Advanced Machine Learning. *IEEE Transactions on Industry Applications*, *61*(2), 2515-2526. https://doi.org/10.1109/TIA.2025.3529804

Neural Networks in RSCAD: Enhancing MMC-Based HVDC Simulation With Advanced Machine Learning

Bara Masalmeh, Rashmi Prasad, *Member, IEEE*, Vaibhav Nougain, *Member, IEEE*, and Aleksandra Lekić, *Senior Member, IEEE*

***Abstract*—The potential of advanced neural networks (NNs) has yet to be explored in the field of HVDC transmission. Implementing such intelligent computational techniques on a real-time digital simulator (RTDS) is challenging due to the need for rapid computation and the risk of overfitting with extensive data generated at tiny time steps. To overcome these limitations, different NN techniques are studied using a supervised and reinforced imitation learning method to mimic the suggested controller with labeled data for real-time applications. Furthermore, the NN component does not necessarily just take a label, and therefore, the authors propose a more advanced approach by incorporating reinforced learning through an error-tracking mechanism into the NN, apart from its loss function. The initial offline processing identifies the best-suited NN technique for online computational feasibility. Both online and offline training methods as well as online adjustments are showcased to provide a robust control solution that is easy to implement. This work deals with developing an intuitive and versatile Toolbox installed on a real-time simulator platform that can integrate complex NN-based control strategies. Extensive simulations on the RTDS platform and experimental investigations of the four terminal HVDC systems validate the interest and viability of the proposed design methodology.**

***Index Terms*—Advanced machine learning, HVDC-based power system, MMC, neural network, real-time EMT simulation, RTDS.**

I. Introduction

THE rapid advancement in control systems employed in inverter-based resources, characterized by dynamics in the order of several kHz, can be most accurately depicted on a real-time simulation platform [2]. This further demands integrating intelligent computational methods into complex systems models in real-time simulations [3]. The application of intelligent computation in power electronics spans various facets, including design optimization, control strategies, and predictive maintenance [4]. Intelligent computation, like neural networks (NNs), excels in tasks where conventional algorithmic approaches may fall short, such as dealing with highly nonlinear systems, managing large-scale data, and adapting to changing system conditions [5]. NNs have shown tremendous potential in handling complex, non-linear, and dynamic systems which can be applied to different power electronics control strategies [6]. Moreover, the capability of adapting to the systems' behavior in real time makes them crucial for the future of smart control of non-linear systems. However, the development of advanced NNs in the real-time-based simulator has been limited by the computational hardware capabilities, the lack of complex programming ability in the software, and the mathematical complexity of the models. Due to this, NNs have only been applied in offline platforms or, when used in real-time, are limited by elementary structures.

A data-driven approach is particularly crucial in scenarios where system dynamics are highly nonlinear and affected by unknown dynamics or parameter uncertainties (due to environment and operating conditions). In this sense, NNs can provide model-free approaches ("method of learning an optimal control policy using process data only") that drop model dependencies and thus address system uncertainties and improve performance are widely established [7]. Some recent studies provide evidence that even drawbacks of advanced control like model predictive control approach can be mitigated with the use of data-driven, NN-based control [8], [9]. Although data-driven techniques have been applied to modular multilevel converters (MMCs) [10], [11], they have not been used extensively in practical systems to obtain measured data from MMC. The main challenge is to reduce the technique's computational requirements to perform the task in real time. An adaptive dynamic programming control strategy, developed from an NN perspective to learn the optimal control of an MMC, is experimentally tested in achieving robust and satisfactory tracking control behaviors in [12]. A data-driven model-free machine learning-enhanced framework is introduced into the inner control strategy of an MMC. This approach can explicitly address uncertainties like unmodeled dynamics and external disturbances, enhancing system robustness [13]. However, the method's complexity increases the computational demand significantly. To address this, our work further proposes a machine-learning-based imitation technique to reduce the

Received 6 June 2024; revised 13 September 2024; accepted 27 November 2024. Date of publication 13 January 2025; date of current version 4 April 2025. Paper 2024-CDPA-0519.R1, approved for publication in the IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS by the Convergence of Data-driven and Physics-based Approaches in Power System Analysis, Optimization, and Control of the IEEE Industry Applications Society. The toolbox created for this work is available in [DOI: 10.5281/zenodo.14742236]. This work was supported by NWO Veni Project SAFE-GRID through Project 20248. *(Corresponding author: Aleksandra Lekić.)*

The authors are with the Faculty of EEMCS, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: A.Lekic@tudelft.nl).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIA.2025.3529804.

Digital Object Identifier 10.1109/TIA.2025.3529804

online computational burden, effectively transforming it into an offline computation process. The methodology demonstrates the potential for real-time complex control strategies, offering a new direction for future research in control methods.

To enable real-time adaptation of voltage balancing in the sub-modules (SMs) of MMCs, the authors in [14] propose using a mixed Gaussian distribution to estimate and compensate for voltage deviations. Additionally, they predict the SM triggering sequence through NNs. This method enhances the precision and speed of voltage balancing while reducing computational complexity, offering a promising solution for managing MMCs more efficiently and reliably. A machine learning-based emulation approach is proposed to reduce the computational load of model predictive control for MMCs by training an artificial NN with data from traditional model predictive control methods. This approach offloads the computational burden from online to offline processes [15]. This greatly reduces the computational time in transient stability simulations of large-scale AC-DC power systems. The study achieves faster-than-real-time digital twin emulation by implementing machine learning-based synchronous generator models and dynamic equivalent models on field programmable gate arrays (FPGAs), with gated recurrent unit algorithms for modeling that enhance accuracy while managing computational complexity. This approach allows for efficient and accurate dynamic security assessment, offering a significant improvement over traditional simulation methods in terms of speed and scalability [7]. There has also been research on the NN implementation in real time digital simulator (RTDS) through simple artificial and recurrent NNs. However, the library is limited to a single layer [16]. In the above work, the challenges posed by power electronic systems (e.g., high tuning speed in control) require specific features in these NNs that differ from other engineering fields, such as image classification. Therefore, further research on adapting to system behavior in real-time is crucial for the future of smart control in non-linear systems.

Nonetheless, recent progress in real-time simulators, such as enhancements in hardware like the introduction of Novacor processors, has significantly improved the computational software capabilities [17]. More complex structures are now possible by utilizing the *CBuilder* interface found in the RTDS software RSCAD. Despite its constraints in *C* compilation due to the irregular processor employed, the authors showcase the utilization of low-level *C* language and several mathematical techniques enabling the execution of sophisticated NNs in RSCAD.

The main contribution of this work is the RSCAD Toolbox for online and offline training, with the development of a multi-layer NN library in RSCAD available in [1]. For the online training component, a Library is created for an artificial NN (ANN) and long short-term memory (LSTM) with multiple layers that are used for training in real time using RTDS. The offline training script file is scripted to consider collecting various case scenarios. Advanced time series NNs such as LSTMs are used to predict the state of the MMC. It is also possible to select parameters and hyperparameters of NN offline with fine-tuning in the online environment.

In the design of the toolbox, we summarize the following contributions:

- An extensive analysis of different NN topologies is presented and trained with data collected from real-time simulator RTDS. An NN library is then designed to run on RSCAD through the *CBuilder* interface to test the performance of the NNs in real-time simulation.
- An adaptive proportional-integral (PI) control using reinforced learning is developed, which helps in the adaptive change of PI tuning parameters with the change in system parameters and configuration. This reduces the need for heuristics in conventional static PI controllers.
- The NN-based inner loop controller is integrated with the outer loop PI controller in a cascaded dual loop control to study the effectiveness of the proposed toolbox. Further, both the loops are replaced by the devised NN controller and with different possible combinations between ANN, LSTM, PI, and adaptive PI.

Although the components built are independent of the use case, two use cases of such networks emerge as relevant in large-scale HVDC-based transmission grids. To the authors' knowledge, this is the first work where such a control real-time toolbox is applied to large-scale MMC controllers for high-power HVDC transmission. The development of a multilayer NN library in RTDS [1] is unique and is not currently present in the literature.

The rest of the article is organized as follows: Section II details the system configuration of the HVDC system and RSCAD/RTDS parameters. Section III discusses the challenges and prospects for NN formulation for real-time control and describes the steps for the NN implementation application in HVDC control in RSCAD. Section IV proposes a Data-Driven Model-free Control technique for Converter behavior emulation and robust predictive control. Section V presents the results, validation, and discussion, whereas section VI provides meaningful conclusions.

II. System Configuration of HVDC-Based Power System

The MMC depicted in Fig. 1 is realized using half-bridge SMs. Each MMC has three legs, and each leg consists of two arms. Each arm of the MMC has N_{SM} SMs. The variables shown in Fig. 1 are defined for all three phases, i.e., $j \in \{a, b, c\}$. Half-bridge SMs are represented by their averaged equivalents, with R_{arm} and L_{arm} being the resistance and inductance, respectively. Each SM has capacitance C_{SM}. The converter model is developed using the $\Sigma - \Delta$ nomenclature. The variables in the upper and lower converter arms can be represented as [18]:

$$i_j^{\Delta} = i_j^U - i_j^L, \quad i_j^{\Sigma} = \frac{i_j^U + i_j^L}{2}, \tag{1}$$

$$v_{Mj}^{\Delta} = \frac{-v_{Mj}^U + v_{Mj}^L}{2}, \quad v_{Mj}^{\Sigma} = \frac{v_{Mj}^U + v_{Mj}^L}{2}. \tag{2}$$

To show the effectiveness and stability of the proposed advanced NNs, they are applied to control a four-terminal half-bridge MMC-based HVDC power system using two offshore wind farms generating a total power of 4GWs, depicted as CSA2 and CSA3 in Fig. 2. In contrast, two onshore grid-connected

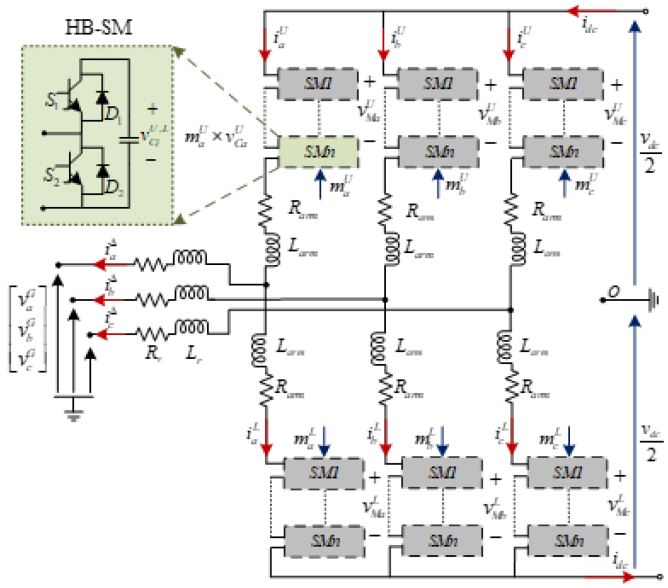

Fig. 1. MMC topology.

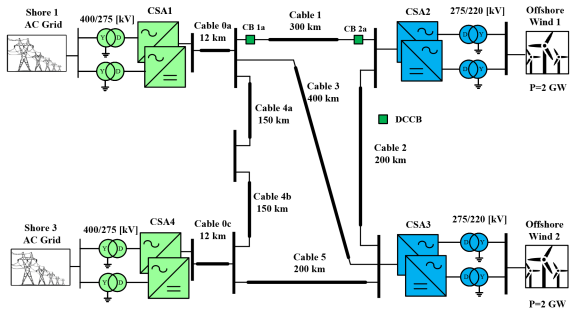

Fig. 2. RSCAD four-terminal HVDC power system [20].

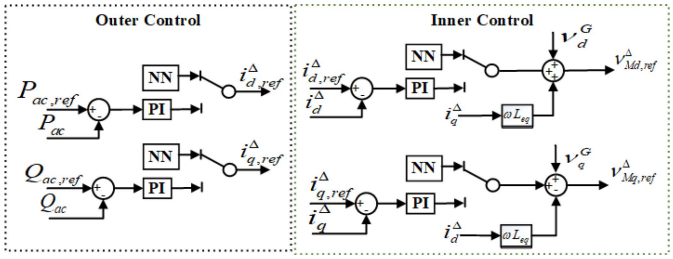

Fig. 3. Control strategies for outer and inner control, selecting between PI and *CBuilder* NN block.

converters (CSA1 and CSA4) are used to transfer this generated power. Frequency-dependent long underground cables in bipolar configuration are used for this power transmission. The two offshore converters connected to the wind farms are grid-forming, while the two onshore converters connected to the grid are grid-following. One of these two converters (given as CSA1) is independently used to maintain the DC voltage of the system (i.e., ±525 kV) using the conventional PI control action. In contrast, the other converter (given as CSA4) is used in active power reference mode [19], [20] using the advanced NNs as shown in Fig. 3.

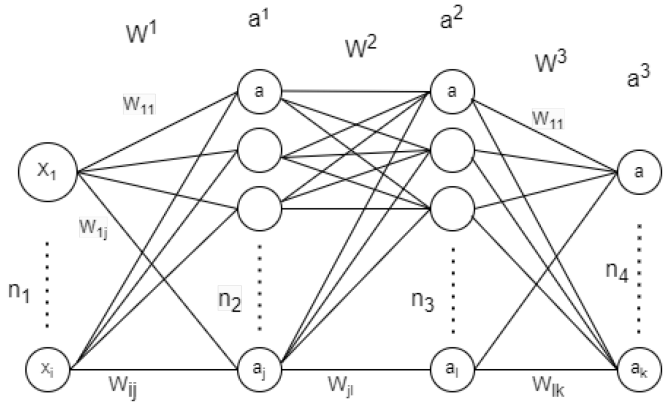

Fig. 4. ANN design.

The MMC control is divided into two categories. The voltage, power, and energy controllers, together with AC and DC currents control, are known as the higher-level control layer, and the low-level control layer is responsible for balancing the SMs and practical aspects of firing pulses generation. This work focuses on higher-level control of the MMC, split into two loops: the outer loop, which generates a current reference signal as an output, and the inner loop, which generates a modulation reference as an output. The proposed technique replaces the dual-loop higher PI controls with NN techniques such as ANN and LSTM.

III. NN Formulation for Real-Time Control

In this work, ANN depicted in Fig. 4 for the layer l (layers values being $l = 0$ for the input, $l = 1$ for the first hidden layer, etc.) is given as:

$$z^{[l]} = W^{[l]} \odot a^{[l-1]} + b^{[l]}, \tag{3a}$$

$$a^{[l]} = \begin{cases} x, & \text{for input to the ANN, } l = 0, \\ \beta(z^{[l]}), & l \in \{1, \ldots, L\}, \end{cases} \tag{3b}$$

for n_l being the number of nodes in layer l, and n_{l-1} number of nodes for the layer $l-1$, etc. and L being the number of layers (e.g. on Fig. 4 number of layers is $L = 3$). Vector $z^{[l-1]} \in \mathcal{R}^{n_{l-1}}$ is the input to the layer l, $b^{[l]} \in \mathcal{R}^{n_l}$ is the vector of biases, and $W \in \mathcal{R}^{n_l \times n_{l-1}}$ is matrix of weights. Function $\beta(\cdot)$ represents the activation function, and $\odot$ denotes the Hadamard product.

The gradient descend optimization algorithm is used to optimize/tune the values of the weights by extrapolating their gradient relative to a loss function and moving in the loss minimization direction for ANN's training following the Chain rule:

$$\frac{\partial L}{\partial W_{ij}^{[l]}} = \frac{\partial L}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial W_{ij}}, \tag{4}$$

where L denotes the defined loss function.

Without memory units, the network treats each time step independently, significantly reducing performance for time series data. Due to the necessity of taking a certain time window of data into account when dealing with time series analysis,

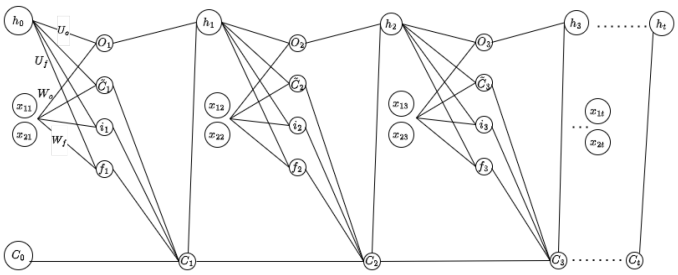

Fig. 5. LSTM design.

different Recurrent NNs were also formulated and tested for the proposed *CBuilder* Component/Toolbox. The most significant of these types is the LSTM NN, depicted in Fig. 5, which its two main variables can summarize at each time step t. These are the hidden state h_t and cell state C_t [21] (can also be referred to as short-term memory and long-term memory variables, respectively):

$$C_t = \sigma\left(U_f h_{t-1} + W_f x_t + b_f\right) \odot C_{t-1} + \sigma\left(U_i h_{t-1} + W_i x_t + b_i\right) \odot \tanh\left(U_C h_{t-1} + W_C x_t + b_C\right), \tag{5}$$

$$h_t = \sigma\left(U_o h_{t-1} + W_o x_t + b_o\right) \odot \tanh(C_t), \tag{6}$$

where x_t is the input at timestep t and $U_{f,i,C,o} \in \mathcal{R}^{n\times n}$, $W_{f,i,C,o} \in \mathcal{R}^{n\times m}$ are the weights of the different gates. $\sigma(\cdot)$ is an activation function. The four gates of the LSTM are the input gate, forget gate, candidate memory gate, and output gate, and each is responsible for a value that, when combined through (6), gives the output of the layer which is used as input into the next timestep.

To summarize, LSTM includes short-term and long-term memory variables. The short-term memory variable is the output of the LSTM layer at time step t. For time-step $t = T$ where T is the last time-step in the calculation, the short-term memory calculated represents the final output of the LSTM layer. A long-term variable C_t carries forward through every time step and influences the short-term memory component. C_{t-1} is the previous long-term memory variable; it can be thought of as the percentage of the long-term memory to remember; ironically, it is called the forget gate. Each gate has 2 unique sets of weights, U, which are the gate weights regarding the short-term memory, and W, which are the weights regarding the input.

A. RTDS Toolbox Guidelines

The RTDS is widely used in academia and industry among different real-time simulators. The RISC processor of NovaCor in RTDS consists of ten cores (CPU units), which operate at 3.5GHz each. In the RTDS, there are different simulation time steps. Each modeling time step has its advantages and disadvantages. For every simulation, three components exist in RTDS: the *Network solution*, *Control*, and *Power System*. Among these components, the *Power System* changes with the type of simulation time step model (i.e., super step, main step, etc.). The *control* works in the main step. Thus, the execution of all *control* units has to be done within 80 μs. The control can be split into several parallel paths for a large system. However, this parallel path is restricted to the number of available cores in RTDS.

NNs are modeled from scratch in Python and *C*. The Python models are used for initial testing, while the *C* code is used to create a toolbox using the RTDS *CBuilder* library. These NNs include multi-layered ANNs, multilayered recurrent NNs (RNNs), and LSTMs [22]. Due to the real-time simulation feature, its intensive data generation process, and the sensitivity of power systems to continuous sudden changes, the online training feature of the toolbox needs to be designed carefully such that overfitting is avoided and training is done for short periods during the same training case. To avoid the complexities of online learning, a Python script has been developed to automatically parse data generated from RSCAD using the script file, train the model offline with various optimizations in TensorFlow, and generate the weight and bias matrices. These matrices can then be manually inserted into the *CBuilder* NN component code.

In total, three components have been developed as a part of this toolbox [1]:

- The ANN component includes up to four hidden layers and 100 weights between layers.
- A NN (including LSTM) component that handles the input of up to 10 variables for 20 time steps.
- A layer component that can be used to build an RNN or ANN in RSCAD layer by layer, with no limitation on the number of hidden layers. This component behaves similarly to Tensorflow, where each layer communicates to the next layer for predictions and to the previous layer for loss backpropagation. The layers can even be distributed across multiple processor units in the RTDS system; although not recommended, this allows for very large NNs to be built. This method also allows for different parameters in each layer, such as activations and learning rates.

  Please note that RNNs are used when performing time-series analysis with sequential input-output data because they effectively leverage the sequential nature of the data to identify trends. In the study, sequential data is generated from the RSCAD based on the network operation.

The limitations imposed on all these components can be extended as the computational limit has surprisingly not been reached. All components come with online learning. Several challenges exist in the application of NNs in RSCAD. The main challenges come from the RTDS specifications as follows:

- Execution time of the NN algorithm to be shorter than 50 μs.
- Creation of the algorithm using *CBuilder* and not availability to use dynamic memory assignment in C (e.g., malloc function).
- Possibility of overfitting due to constant streaming of data.
- Delays between software and RSCAD due to communication protocol of $100-200$ μs.

1) Overfitting: The application of unoptimized NN libraries is the tendency of these networks to overfit the presented data. This problem is exaggerated due to the nature of RSCAD, which generates 20000 data points every second (when using

a 50 μs simulation timestep for real-time simulation). This is pronounced, especially at the start of the training process when the untrained model sees many data points for the same output range. The significance of this issue diminishes when the model weights are initialized offline and adjusted during online training. However, the problem can still be mitigated by the use of the batch number, regularization, and learning rate [23], [24] as input parameters to the network. The learning rate is set as a slider, which means it can be changed in real time. A varying learning rate depending on the size and progression of the error ensures the model has a "just right" fit. Furthermore, using low-pass filters reduces the noise of the target output during the training process.

The learning rate dictates the scale of change of the weights due to the loss at each simulation timestep. By applying an adaptive learning rate, the user can employ multiple training algorithms such as 'Adam' and 'RMSprop', where the learning rate is adjusted based on the size and progression of the error [25], [26]. Such algorithms were not employed inside the component to give flexibility to the user, where the learning rate can be manually changed throughout the simulation or by using the discussed algorithm in the real-time scripting feature in RSCAD software.

Regularization is a method used to avoid overfitting by adding a penalty term to the loss function. This encourages sparsity in the model parameters and, in some cases (L1 regularization) can lead to the dropping of certain weights. The two types of regularization methods mainly used in NNs are L1 and L2 regularization, defined by the following equations:

$$L_{\mathrm{L1}} = L_0 + \lambda \sum_{i=1}^{n} |w_i|, \quad L_{\mathrm{L2}} = L_0 + \lambda \sum_{i=1}^{n} w_i^2 \qquad (7)$$

where L_0 is the original loss function without regularization, λ is the regularization parameter that controls the strength of the regularization term, w_i represents the coefficients or weights of the model, and n is the number of features or weights in the model. The ability to drop weights out can be beneficial when multiple outputs of the NN do not depend on the same set of inputs. However, applying this regularization in the toolbox came with the challenge of many weights dropping out as RSCAD generates hundreds of thousands of timesteps every minute. Therefore, a new feature is required that can turn on and off this regularization feature. To mitigate this problem, the lambda term has been made adaptive and can be changed throughout the simulation. In this work, therefore, we use L2 regularization.

Finally, the user can use a low pass filter for the loss function calculation as a way to reduce the noise of the target output during the training process, this will lead to a smoother performance by the NN when the learning stops.

2) Dynamic Memory Space: RSCADs inability to allocate dynamic memory space in real-time, which is due to the method of compilation of the *CBuilder* components where a *C* file is turned into an assembly file and an object file before its integration in any simulation model limits how dynamic the interaction between the user and the toolbox can be. *CBuilder* cannot use the standard library in *C*, which has many consequences, such as the inability to use the Malloc function (Method to dynamically allocate memory). This issue is mitigated by defining the NNs with the maximum memory size used. Then, through user-defined input parameters and mathematically isolating a certain part of the network, a smaller NN can be imitated.

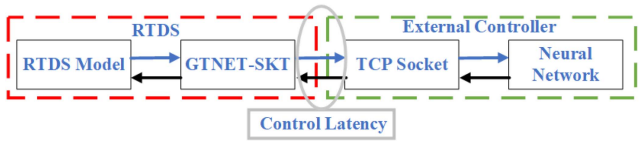

Fig. 6. Schematic representing an established connection between the external machine where NN runs and RTDS where MMC model runs.

B. RTDS Toolbox Library Structure

The RTDS environment interfaces with a GNET-SKT (Generic Network Socket) as a network gateway for data exchange between the simulation and external devices or software. The GNET-SKT uses a TCP/IP socket to send and receive data packets, which are sent to an external controller for input to NNs. This method is used to establish a connection with the running simulation of the network found in Fig. 6. However, the TCP connection introduces control latency, typically between 100 and 200 microseconds, which can be significant for some use cases due to the round-trip data exchange delay. This setup is primarily used for testing real-time prediction and training of NNs before direct implementation in RSCAD. Additionally, RSCAD features a *CBuilder* interface that allows users to design components using low-level *C* programming manually. A NN library can be run on RSCAD through this *CBuilder* interface thus avoiding control latency.

1) CBuilder: To satisfy real-time constraints, the *CBuilder* code must be efficient and cannot take excessively long to execute to avoid triggering a timestep overflow error. The Static and RAM sections are not utilized in building an NN component except for weight and parameter initialization. This is because there are no constants. However, for NNs without an online training functionality, static values and structures can be used for faster computation of more complex NNs. For online training, functionality was successfully added in the CODE_FUNCTIONS section, which consists of user-defined functions as shown in Fig. 7.

The proposed library is flexible and has various input features. Namely, it takes the loss function as one of the inputs, which allows the library to be used for any application, as the variables it trains depend on the specified loss function. Furthermore, we can select the following parameters:

- Backpropagation (Gradient Descend) for tuning the weights;
- Learning Rate - it initially tunes based on the response; and as the NN result converges, it is reduced.
- Inputs are based on physics and control type (also on feature importance);
- The architecture of the NN - in this work, it was based on the simplest NN that achieved a good Mean Squared Error

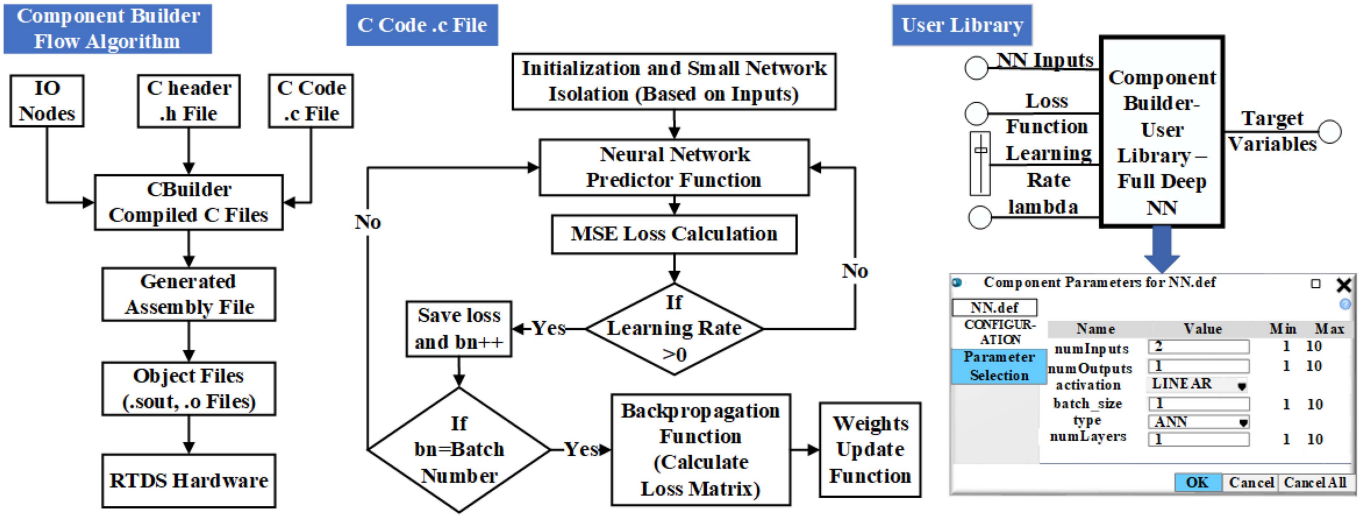

Fig. 7. Simplified *CBuilder* compilation process.

(MSE) score (we did a grid search which tests different acts like weights and layers/activation functions).

The Various combinations of Toolbox Features are included in the full NN and layer component NN library as denoted:

- `numInputs`: Number of features the NN tries to map (1-10).
- `numOutputs`: Number of prediction variables (1-10).
- `type`: Type of NN model (ANN, LSTM, RNN).
- `numLayers`: Number of hidden layers between input and output layer.
- `numNodes`: Number of LSTM nodes (1-20), activated only when type is chosen to be LSTM.
- `numWeights`: Number of Weights in hidden layers (1-100 per layer).
- `timesteps`: Number of previous timesteps taken by the LSTM (1-20).
- `activation`: Activation function in the hidden layers (tanh, linear, ReLu, Sigmoid), not applicable for LSTM models as they have predefined activation.
- `batch_size`: The number of iterations before the backpropagation algorithm is executed.
- `Learning-Rate`: The rate at which NN learns.
- `lambda`: Regularization term to drop out weights.

For the layer component of NN, the user must manually input the target variable and the learning rate for the next layer. The learning rate is taken as a dynamic input, allowing the user to adjust it to find the optimal rate, which determines the extent of weight adjustments during each backpropagation iteration. The *C* association files must be initialized after defining the initial parameters, nodes (inputs/outputs), and graphics. *CBuilder* enables different *C* scripts based on the parameters defined, allowing multiple scripts to be included in one component, with only the relevant script executed based on the selected parameters. The I/O nodes must be defined again in the inputs/outputs tabs for use within the *C* script.

2) Compilation Process: The *CBuilder* compiles all the relevant. h and. c files and generates *C* files of its own, most likely for compatibility with hardware. The files are then further decoded into an assembly file followed by object files which are finally used to construct the User Library. The *CBuilder* flow diagram is shown in Fig. 7. As mentioned before, different combinations of NN can be formulated in *CBuilder*. The extreme right of Fig. 7 shows the User Library of the full NN, in which component parameters of different options of activation functions and NN model can be selected from the drop-down menu. All various option selections are specified in the Component Parameters tab, which must be chosen before RSCAD compilation. However, due to computational constraints, the Full NN model is restricted to a maximum of four layers. Alternatively, the authors introduced another option with additional layers, termed the "layer component," where each layer is defined within the individual User library.

C. Selection of NN Techniques

NN computation in RSCAD must be fast to complete each step execution within 50 μs to avoid time overflow problems. For the selection of model, the different models will be tested on data sets in Python for various time series advanced NN models like ANN, RNN, LSTM, Gate Recurrent NN (GRU), Convolutional NN (CNN), and Convolutional Recurrent NN (CRNN). As shown in Fig. 8, model selection is based on the metrics such as training time (Fig. 8(a)), resource consumption (Fig. 8(b)), and MSE (Fig. 8(c)).

As expected, none of the models struggled with modeling the PI. A simple ANN would suffice, but using an RNN might lead to smoother control action during the transient state. Fig. 8(c) shows how the models gain more knowledge of the system after every Epoch (Offline Training Iteration).

Furthermore, the ability to use parallel programming in Python allows for faster computation of NNs. The extent of which is tested using the LSTM model can be found in Fig. 9, which depicts the prediction speed given variable LSTM layers and timesteps.

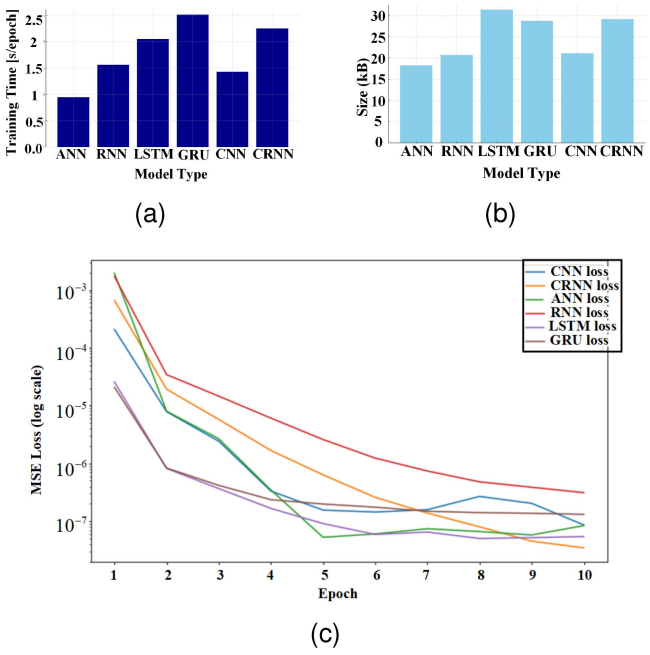

Fig. 8. Model type selection based on (a) training time [s/epoch], (b) memory requirement (kB), (c) the mean squared error (log) loss calculated after every epoch.

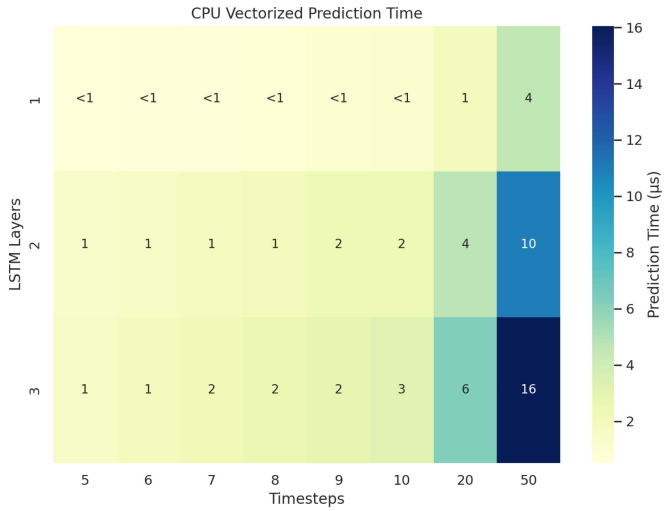

Fig. 9. Prediction time for LSTM per timestep length per number of layers for a vectorized input computed on CPU i7-13k.

For this work, this analysis was carried out to predict in real-time through a TCP connection. From Fig. 9, it can be seen that also the prediction time is lower than 50 μs.

For further studies, the authors have considered ANN and more advanced NN, i.e. LSTM, to showcase various test applications of the designed *CBuilder* library [1] in the next sections.

IV. Proposed Data-Driven Model Free Control in RSCAD

The intricate nature of MMC models demands substantial computational resources. This work explores the integration of NNs into the control architecture of the MMC. First, the NN Toolbox is implemented as discussed in the previous section. Secondly, the loss function is designed based on the optimization problem at hand.

A. NN for Converter Control

The design of the proposed NN control involves integration with the state-of-art PI control by simply mapping the output of the PI control and the input error variables of the NN. Further, stepwise modifications are done in the NN control architecture to make an intuitively better control input along with the inherent advantages of PI control. First, the outer and inner loops of the current control structure can be expressed as:

$$i^{\Delta}_{dq,ref} = PI((P,Q)_{err}), \tag{8a}$$

$$v^{\Delta}_{mdq} = v^{G}_{dq} \pm \omega L i^{\Delta}_{qd} + PI(i_{dq,err}), \tag{8b}$$

where the PI term can be expressed as:

$$\mathrm{PI}(t) = K_p \cdot \mathrm{err}(t) + K_i \cdot \int \mathrm{err}(t)\, dt. \tag{9}$$

At the time t, $\mathrm{PI}(t)$ is the controller output, $\mathrm{err}(t)$ is the error signal (P_{err}/Q_{err} and $i_{d,err}$/$i_{q,err}$), which is the difference between the desired setpoint and the process variable. K_p and K_i are the proportional and integral gains, respectively. Furthermore, they are a tuning parameter that determines the reaction to the current error and a tuning parameter that determines the reaction based on the accumulation of past errors.

The NN components can be implemented in the loop by replacing the PI term or the entire loop. This is done by defining NN with a loss function that finds the path of error minimization. As a result, the idea of stepwise modification is incorporated into the NN control design. Incrementally, the input error function is refined with different approaches given ahead in the subsection.

An NN can be trained offline to extrapolate the value of the control action by mapping input variables $\mathrm{err}(t)$ to the $PI(t)$ output. In this case, the loss used to train the NN can be taken as the MSE loss:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{10}$$

Here, the NN trains based on the direct value the PI gives. However, since, in this case, the target y values would be the PI control action, the NN would train based on the PI control action, basically mimicking the PI. The NN would need an addition to the loss function to perform better than the PI. This can be done by taking into account a second loss term in the expression that takes into account the remaining loss after PI control action (the PI shortcoming), and this can be expressed as:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda(\mathrm{err}(t+1)), \tag{11}$$

where $\mathrm{err}(t+1)$ represents the next time step error due to the control action y_i, and $\lambda > 0$ is a scaling constant. The approach described here tries to train the NN not only based on the corresponding PI action, but also including the remaining error after the PI action.

The authors propose a more advanced approach by incorporating *reinforced learning* through an error-tracking mechanism into the NN, separate from its loss function. This mechanism is activated after the NN completes training and is used for online prediction. In this approach, the weights and biases of the network continuously update during its prediction operation achieving a robust control of the MMC. To implement such a *reinforced learning feature*, a loss function needs to be constructed based on the relationship between the NN output (control action) and the next error such that:

$$Loss = \text{err}(t+1). \tag{12}$$

In this approach, the NN continuously updates the weights after its training process to continuously minimize the error independent of any relationship to the PI controller.

To adapt the weights of the NN to minimize the loss, the following gradient will be needed:

$$\frac{\delta Loss}{\delta \hat{y}(t)} = \frac{\delta \,\text{err}(t+1)}{\delta \hat{y}(t)}. \tag{13}$$

Different methods for solving this (13); however, for a model-free approach, the authors suggest taking the numerical approximation:

$$\frac{\delta \,\text{err}(t+1)}{\delta \hat{y}(t)} = \frac{\text{err}(t+1)_{t+1} - \text{err}(t+1)_t}{\hat{y}(t)_{t+1} - \hat{y}(t)_t}. \tag{14}$$

However, due to the small values involved, this method might be unstable. To overcome the instability of (14) the authors suggest using the same numerical approximation found in (14) but for a loss function that includes the square sum of future loss over a certain time period T such that:

$$Loss(t) = \frac{1}{T}\sum_{k=t-T+1}^{t+1} \text{err}(k)^2. \tag{15}$$

The next time step error can be used in these equations since the weight updates can be postponed till the next error is determined. The gradient descend training algorithm is implemented at the beginning of the timestep based on the previous output and the current error.

To summarize the approach, the NN can initially train based on the PI control action as well as the remaining error after the PI action both incorporated in the loss function that updates the weights of the network as given in (11). When the PI is disconnected and the NN can no longer update its weights based on the loss function found in (11), the weights are updated based on a loss function that aims to minimize the error following each control signal as seen in (12), this makes the NN more robust and adaptive.

B. Adaptive PI Using Reinforced Learning

For adaptive PI tuning, the authors propose a similar approach to updating NN parameters over time using *reinforced learning*. The PI parameters can be adjusted based on the same gradient descent approach [27], [28]. The PI controller equation is given by (9). To make the PI controller adaptive, we use weights W_p and W_i instead of fixed gains:

TABLE I
CONVERTER VARIABLES

Feature	Definition
i_d^{Δ}	Current Component in direct (d) axis in dq-frame
v_d^G	Voltage along the d-axis
i_q^{Δ}	Current component in the q-axis
v_q^G	Voltage along the q-axis
P_{ac}	Converter active power
Q_{ac}	Converter reactive power
v_{dc}, i_{dc}	Voltage and current at DC side
P_{err}/Q_{err}	Error signals (ref values) for outer loop
$i_{d,err}/i_{q,err}$	Error signals (ref values) for inner loop
v_{md}, v_{mq}	Modulation indices reference
$i_{d,ref}^{\Delta}, i_{q,ref}^{\Delta}$	State variable references

$$\text{PI}(t) = W_p(t)\cdot \text{err}(t) + W_i(t)\cdot \int \text{err}(t)\,dt. \tag{16}$$

The weights W_p and W_i are updated using gradient descent to minimize a loss function $L(t)$, which is defined as the MSE over a time window size T:

$$L(t) = \frac{1}{T}\sum_{k=t-T+1}^{t+1} \text{err}(k)^2. \tag{17}$$

The gradient of the loss function concerning the weights is calculated as follows:

$$\frac{\partial L(t)}{\partial W_{p,i}} = \frac{2}{T}\sum_{k=t-T+1}^{t+1} \text{err}(k)\cdot \frac{\partial \text{err}(k)}{\partial W_{p,i}} \tag{18}$$

Assuming the error at time step $t+1$ is only affected by the PI output at time t: $\frac{\partial \text{err}(t+1)}{\partial \text{PI}(t)} \approx -1$, the partial derivatives of the error concerning the weights can be approximated as:

$$\frac{\partial \text{err}(k)}{\partial W_p} \approx -\text{err}(k), \quad \frac{\partial \text{err}(k)}{\partial W_i} \approx -\int \text{err}(k-1)\,dt. \tag{19}$$

Substituting these approximations into the gradient equations:

$$\frac{\partial L(t)}{\partial W_p} = -\frac{2}{T}\sum_{k=t-T+1}^{t} \text{err}(k)^2, \tag{20a}$$

$$\frac{\partial L(t)}{\partial W_i} = -\frac{2}{T}\sum_{k=t-T+1}^{t} \text{err}(k)\cdot \int \text{err}(k-1)\,dt, \tag{20b}$$

the weights are updated using gradient descent:

$$W_{p,i}(t+1) = W_{p,i}(t) - \eta_{p,i}\frac{\partial L(t)}{\partial W_{p,i}}, \tag{21}$$

where η_p and η_i are the learning rates for W_p and W_i, respectively. Subscript p and i denote the equations and variables for proportional and integral gains, respectively.

In summary, the adaptive PI control method continuously adjusts the weights W_p and W_i to minimize the loss function, improving the system's performance over time.

TABLE II
CONTROL MODE AND PARAMETER OF MMC-HVDC SYSTEM

Parameter	Converters			
	CSA 1	CSA2	CSA3	CSA4
Active power [MW]	(4000-Pref)	2000	2000	Pref
Control mode	Vdc-Vac	Vac-f	Vac-f	Pref-Vac
Reactive power [MVAR]	-	-	-	-
Number of Submodules per arm	240	200	200	240
MMC arm inductance [mH]	25	49.7	49.7	25
Transformer leakage reactance [pu]	0.18	0.15	0.15	0.18
AC Point of couple voltage [kV]	400	220	220	400
DC link Voltage [kV]	±525			
Rated power [MVA]	2000			
Capacitor energy in a submodule (SM) [MJ]	30			
MMC arm resistance [Ω]	0.544			
Rated voltage/current of each SM [kV/kA]	2.5/2			
AC converter bus voltage [kV]	275			
Wind farm AC Point of couple voltage [kV]	66			

V. RESULTS AND DISCUSSIONS

The proposed real-time *CBuilder* toolbox application [1] is demonstrated through the following test cases in the four-terminal MMC-based HVDC power system with parameters with denoting conventions as given in Table I, whose values are indicated in Table II for:

- converter control emulation with different control structures in CSA4 - (i) replacing outer and inner current control loops with various combinations of NNs and PI, (ii) comparison on ANN and LSTM models with PI controller;
- an adaptive PI for fine-tuning method showcasing the ability of NNs to predict the operation of the MMC based on its previous states.

The learning rate of NN is adjusted by monitoring the behavior of the curve. For example, during training, if the direction of prediction and the direction of the target are not similar during a sudden change in input parameters, then the weights do not have the correct signs. In this case, the learning rate will be set high at $0.0001 - 0.01$ until the NN output and the target variable move in the same direction during changes in input parameters. Once that is achieved, the learning rate is then reduced to somewhere $10^{-9} - 10^{-4}$. It is ensured not to keep the same input parameter range for extended durations of time. If, during training, the shape of the prediction during transient is very odd, then the training is stopped by setting the learning rate to zero during the transient and it is resumed after transient. For the regularization term, λ, a value of 0.0 can be taken to avoid complications. However, if necessary, then the value of λ is set as a small number (e.g., 10^{-8}) and it is adjusted accordingly and reduced to 0 after a certain time period. Setting λ to a larger value (e.g., $\geq 10^{-4}$) can lead to most of the weights dropping out, an appropriate value of lambda most likely is found in the range $10^{-8} - 10^{-4}$, the exact number would be dependent on how long this regularization is applied.

A. Comparison of Converter Control Approaches

The results in this section are presented for an ANN and LSTM as it replaces a PI controller in generating a control action signal for the inner and outer loops of the controller. This is done for the AC-side control (i_q loops) and the DC-side voltage or active power control (i_d loops for cases where the reactive current contribution is absent). The test case includes a transient operation period where the active power reference is abruptly changed from -500 MW to 500 MW. This physically means that the converter goes from moving 1000 MW of power to the AC grid to providing 1000 MW to the DC grid instead. The difference in reference values to actual values is due to the converter's positive/negative sequence design. Due to the error tracking technique integration into the NN, the weights are based on the current error and the previous prediction. The current and voltage measurements used in the modeling are processed with a second-order low-pass filter, which provides robustness against measurement noise in the signals. The proposed control method is focussed on small-signal stability and does not cover large-signal stability such as fault transients, $N-1$ transients, etc.

To understand the influence of the control approach on the system, the converters' state variables are shown in Fig. 10. Results show a quite promising response which depicts a faster response speed by the LSTM when implemented on both outer and inner loops; this is expected as, unlike the PI, the network acts instantly to the large change in the error signal. Also, unlike using a proportional controller, the control action does not overshoot and settles quite quickly. Ideally, the *dq* components are decoupled, easing the separate control of the AC and DC side; however, during the transient state, certain dynamics can lead to the coupling of the two terms, as shown by the response of the q-axis components of the PI controller to an abrupt change in active power. The response of the NN for Q_{ac}, i_q^{Δ}, and $i_{q,ref}^{\Delta}$ shows higher dynamics due to the influence of coupling but still has a faster response. While the response of the P_{ac}, i_d^{Δ}, and $i_{d,ref}^{\Delta}$ is quicker and damps better disturbances than the PI response, which is important for keeping the overshoots bellow the maximum allowed values needed for the proper operations of the HVDC-based power system. This case demonstrates response speed improvement by the NN technique for both the inner and outer loop.

Fig. 11 compares the performance of ANN and LSTM with PI control, demonstrating a smooth settling to active power without damping or overshooting. The ANN, lacking the ability to propagate through time, may struggle to enhance the controller's overall performance, as shown in the figure where the ANN's response resembles that of the PI controller. However, an LSTM would theoretically have such capabilities as it could derive steady state errors through the progression of the error and state variables over time. However, such technicality is not expected to weigh heavily since, for the outer loop, a P controller is usually sufficient. Moreover, the LSTM might also benefit from a limitation imposed on the rate of change in control action due to the initial peak caused by the large error signal. Overall, all controllers maintain stability during the transient state with the LSTM model having the smoothest response and the ANN model having the faster response.

B. Adaptive PI

The test scenarios were made by evaluating the performance of the PI when the integral and proportional parameters are given

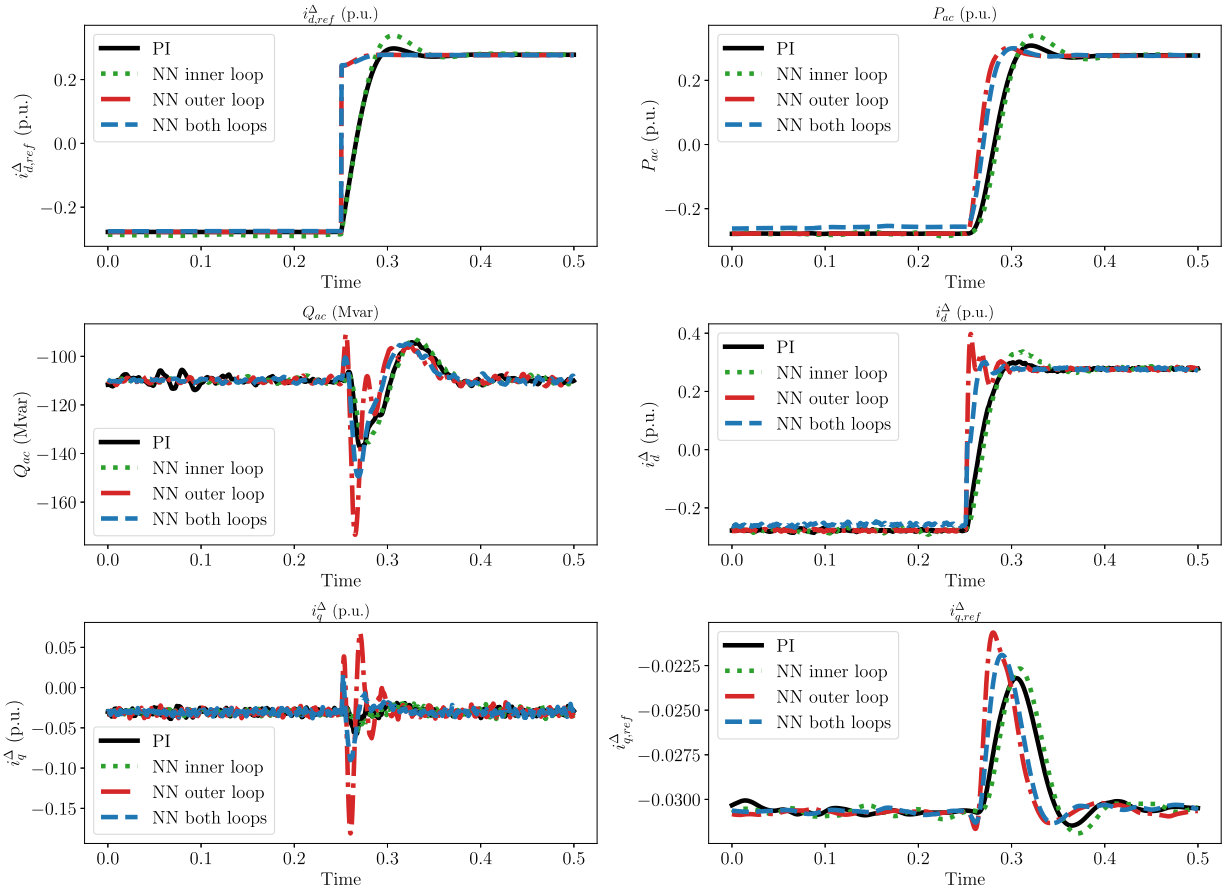

Fig. 10. Comparison of reaction of state variables $i^{\Delta}_{d,q}$, their reference values and active and reactive power on the step change in the active power reference from −0.25 p.u. to 0.25 p.u. When the control is established for PI in both inner and outer control loops - solid black line denoted as PI; LSTM in the inner loop and PI in outer - dotted green light denoted as an inner loop; LSTM in outer and PI in the inner loop - read line denoted as an outer loop; And LSTM in both loops - dashed blue line denoted as both loops.

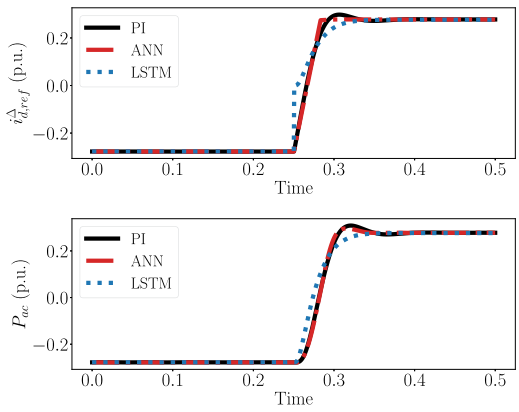

Fig. 11. Comparison of reaction of state variable i^{Δ}_{d} and active power on the step change in the active power reference from −0.25 p.u. to 0.25 p.u. when the control is established for PI in both inner and outer control loops - solid black line denoted as PI; ANN in both inner and outer control loops - solid red line denoted as ANN and LSTM in both inner and outer control loops - dotted blue line denoted as LSTM.

an offset. The adaptive PI mechanism was then implemented and tested directly after implementation and also tested after the PI terms settled to their new values. The inclusion of adaptive online learning enabled the NN models to maintain optimal performance even with short training periods. By incorporating the error term into the loss function, the NN was able to improve and learn without relying on a PI controller for online training.

Fig. 12 shows the response of the classical PI, which is compared with the proposed technique with the adaptive technique after waiting for it to tune itself and is also compared against without waiting for tuning. It can be observed from the figures that the proposed adaptive PI technique, both trained and even with training, performs better than conventional PI response. In the case of the adaptive PI control, the tuning parameters adaptively converge to their new operation point upon a transient. This eliminates the need for heuristics related to PI tuning, which conventionally is challenging. The convergence of adaptive PI tuning gains leads to faster transient response of the controller along with the attribute of robust steady state performance, which is the inherent property of PI controller.

Focussing on the limitations of the controller in comparison to conventional PI, adaptive PI takes more computation than conventional PI which is simpler to implement. Adaptive PI brings the attribute of superior transient performance at the cost of computation to tune the parameters according to the system changes and parameter variation.

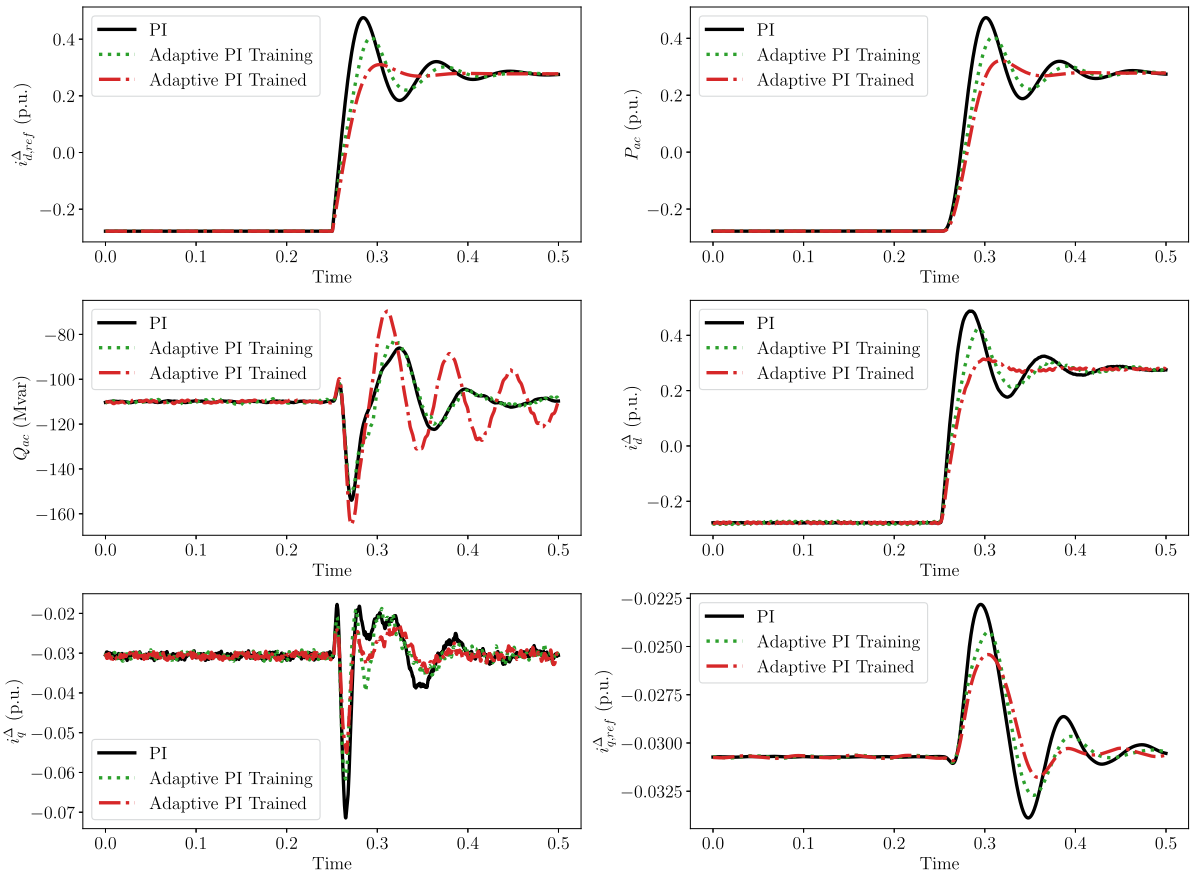

Fig. 12. Comparison of reaction of state variables $i^{\Delta}_{d,q}$, their reference values and active and reactive power on the step change in the active power reference from -0.25 p.u. to 0.25 p.u. when the control is established for PI in both inner and outer control loops - solid black line denoted as PI; Adaptive PI in outer and inner loop with ongoing training - dotted green light denoted as adaptive PI training; Adaptive PI in outer and inner loop with trained parameters - dashed red line denoted as adaptive PI trained.

VI. Conclusion

This paper proposes a resilient, adaptive model-free feature for the standard cascaded PI control strategy. This represents a unique implementation of advanced NN control techniques in a real-time simulator, demonstrating stability even in large-scale test cases showcased on large-scale MMC controllers for high-power HVDC transmission systems. The developed RSCAD Toolbox with a multi-layer NN library for online and offline training is unique and the first of a kind. It offers the following functionalities:

- Possibility of offline, online, or combined training of the NN components.
- For the online training component, an NN Library is created for ANNs and RNNs with 1,2,3, or more layers, all with real-time training capabilities.
- The toolbox allows full flexibility in defining the structure of the NN and the learning method where any loss function defined in RSCAD can then be used as the training loss for any of the NNs.
- The toolbox also allows flexibility for choosing a number of controlled parameters and allows the use of the training for control of both inner and outer control loops or just one of them.

Such a toolbox opens the door to many possibilities for the real-time testing of NN applications in HVDC systems.

In this paper, we have shown that real-time tuning of the PI controller using this NN toolbox in RTDS shows better control response, especially in cases when there is a disturbance in the HVDC power system, such as active and reactive power change.

References

[1] B. Masalmeh, R. Prasad, V. Nougain, and A. Lekić, "RTDs neural network models," (n.d.), 2025, doi: 10.5281/zenodo.14742236. [Online]. Available: https://github.com/control-protection-grids-tudelft/RTDS_NN_models

[2] Y. Lin et al., "Pathways to the next-generation power system with inverter-based resources: Challenges and recommendations," *IEEE Electrific. Mag.*, vol. 10, no. 1, pp. 10–21, Mar. 2022.

[3] M. K. Singh, S. Gupta, and V. Kekatos, "Machine learning for optimal inverter operation in distribution grids," in *Proc. 55th Annu. Conf. Inf. Sci. Syst.*, 2021, pp. 1–1.

[4] S. Zhao, F. Blaabjerg, and H. Wang, "An overview of artificial intelligence applications for power electronics," *IEEE Trans. Power Electron.*, vol. 36, no. 4, pp. 4633–4658, Apr. 2021.

[5] S. Wang, T. Dragicevic, Y. Gao, and R. Teodorescu, "Neural network based model predictive controllers for modular multilevel converters," *IEEE Trans. Energy Convers.*, vol. 36, no. 2, pp. 1562–1571, Jun. 2021.

[6] M. Massaoudi, H. Abu-Rub, S. S. Refaat, I. Chihi, and F. S. Oueslati, "Deep learning in smart grid technology: A review of recent advancements and future prospects," *IEEE Access*, vol. 9, pp. 54558–54578, 2021.

[7] S. Cao, V. Dinavahi, and N. Lin, "Machine learning based transient stability emulation and dynamic system equivalencing of large-scale AC-DC grids for faster-than-real-time digital twin," *IEEE Access*, vol. 10, pp. 112975–112988, 2022.
[8] X. Liu et al., "Data-driven neural predictors-based robust MPC for power converters," *IEEE Trans. Power Electron.*, vol. 37, no. 10, pp. 11650–11661, Oct. 2022.
[9] W. Wu et al., "Data-driven iterative learning predictive control for power converters," *IEEE Trans. Power Electron.*, vol. 37, no. 12, pp. 14028–14033, Dec. 2022.
[10] F. Salazar-Caceres, M. Bueno-López, and S. Sanchez-Acevedo, "Contraction-based controller for a MMC based on data-driven nonlinear identification with noisy data," in *Proc. IEEE 5th Colombian Conf. Autom. Control*, 2021, pp. 216–221.
[11] W. Wu et al., "Model-free sequential predictive control for MMC with variable candidate set," *IEEE J. Emerg. Sel. Topics Power Electron.*, vol. 11, no. 3, pp. 2942–2951, Jun. 2023.
[12] X. Liu, L. Qiu, J. Rodríguez, K. Wang, Y. Li, and Y. Fang, "Learning-based neural dynamic surface predictive control for MMC," *IEEE Trans. Power Electron.*, vol. 38, no. 1, pp. 53–59, Jan. 2023.
[13] X. Liu, L. Qiu, Y. Fang, and J. Rodríguez, "Predictor-based data-driven model-free adaptive predictive control of power converters using machine learning," *IEEE Trans. Ind. Electron.*, vol. 70, no. 8, pp. 7591–7603, Aug. 2023.
[14] Q. Xi, Y. Tian, and Y. Fan, "Capacitor voltage balancing control of MMC sub-module based on neural network prediction," *Electronics*, vol. 13, no. 4, Feb. 2024, Art. no. 795.
[15] S. Wang, T. Dragicevic, G. F. Gontijo, S. K. Chaudhary, and R. Teodorescu, "Machine learning emulation of model predictive control for modular multilevel converters," *IEEE Trans. Ind. Electron.*, vol. 68, no. 11, pp. 11628–11634, Nov. 2021.
[16] B. Luitel, G. K. Venayagamoorthy, and G. Oliveira, "Developing neural networks library in RSCAD for real-time power system simulation," in *Proc. 2013 IEEE Comput. Intell. Appl. Smart Grid*, 2013, pp. 130–137.
[17] K. Sidwall, "The closed-loop revolution," *IEEE Power Energy Mag.*, vol. 18, no. 2, pp. 38–46, Mar./Apr. 2020.
[18] A. Lekić and J. Beerten, "Generalized multiport representation of power systems using abcd parameters for harmonic stability analysis," *Electric Power Syst. Res.*, vol. 189, 2020, Art. no. 106658.
[19] L. Liu, A. Shetgaonkar, and A. Lekić, "Interoperability of classical and advanced controllers in MMC based MTDC power system," *Int. J. Elect. Power Energy Syst.*, vol. 148, 2023, Art. no. 108980. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061523000376
[20] A. Shetgaonkar, T. Karmokar, M. Popov, and A. Lekić, "Enhanced real-time multi-terminal HVDC power system benchmark models with performance evaluation strategies," *CIGRE Sci. Eng.*, vol. 32, pp. 1–29, 2024.
[21] D. M. Stipanović et al., "Some local stability properties of an autonomous long short-term memory neural network model," in *Proc. 2018 IEEE Int. Symp. Circuits Syst.*, 2018, pp. 1–5.
[22] P. Qashqai, K. Al-Haddad, and R. Zgheib, "Modeling power electronic converters using a method based on long-short term memory (LSTM) networks," in *Proc. IECON 46th Annu. Conf. IEEE Ind. Electron. Soc.*, 2020, pp. 4697–4702.
[23] J. Ba and B. Frey, "Adaptive dropout for training deep neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2013, vol. 26, pp. 1–9.
[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT press, 2016.
[25] D. P. Kingma "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[26] T. Tieleman and G. L. HintonHinton, "6.5-RMSProp," COURSERA: Neural Netw. Mach. Learn., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., vol. 6, 2012. Accessed: June 30, 2024. [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
[27] B. Derajić, I. Krčmar, P. Marić, P. Matić, and D. Marčetić, "A normalised gradient descent PI controller for speed servomechanism," in *Proc. 21st Int. Symp. INFOTEH-JAHORINA*, 2022, pp. 1–6.
[28] C. Bruce, "PID controller optimization: A gradient descent approach — towardsdatascience.com," 2024. Accessed: Apr. 09, 2024. [Online]. Available: https://towardsdatascience.com/pid-controller-optimization-a-gradient-descent-approach-58876e14eef2