UNIVERSIDADE FEDERAL DE MINAS GERAIS
SCHOOL OF ENGINEERING
GRADUATE PROGRAM IN MECHANICAL ENGINEERING

RAMON DE PAOLI MENDES

TEMPERATURE CONTROL IN AN AIR CONDITIONING SYSTEM THROUGH CLOUD-BASED DEEP LEARNING ALGORITHMS FOR ADAPTIVE TUNING OF THE PI CONTROLLER

Belo Horizonte
2024

RAMON DE PAOLI MENDES

TEMPERATURE CONTROL IN AN AIR CONDITIONING SYSTEM THROUGH CLOUD-BASED DEEP LEARNING ALGORITHMS FOR ADAPTIVE TUNING OF THE PI CONTROLLER

Thesis presented to the Graduate Program in Mechanical Engineering at the Federal University of Minas Gerais as a partial requirement for the attainment of the Doctoral degree in Mechanical Engeering

Advisor: Prof Drº Luiz Machado

Co-Advisor: Prof Drº Juan José Garcia Pabón

Belo Horizonte
2024

Mendes, Ramon de Paoli.
M538t Temperature control in an air conditioning system through cloud-based deep learning algorithms for adaptive tuning of the PI controller / Ramon de Paoli Mendes. - 2024.
1 recurso online (118 f. : il., color.) : pdf.

Orientador: Luiz Machado.
Coorientador: Juan José Garcia Pabón.

Tese (doutorado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Anexos: f. 96-118.

Bibliografia: f. 87-95.

1. Engenharia mecânica - Teses. 2. Aprendizado profundo - Teses.. 3. Controle de temperatura - Teses. 4. Inteligência artificial - Teses. 5. Algoritmos - Teses. I. Machado,Luiz. II. Garcia Pabón, Juan José. III. Universidade Federal de Minas Gerais. Escola de Engenharia. IV. Título.

CDU: 621(043)

Ficha catalográfica elaborada pelo bibliotecário Reginaldo César Vital dos Santos CRB/6 2165
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

FOLHA DE APROVAÇÃO

"TEMPERATURE CONTROL IN AN AIR CONDITIONING SYSTEM THROUGH CLOUD-BASED DEEP LEARNING ALGORITHMS FOR ADAPTIVE TUNING OF THE PI CONTROLLER"

RAMON DE PAOLI MENDES

Tese submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Minas Gerais, constituída pelos Professores: Dr. Luiz Machado (orientador Departamento de Engenharia Mecânica - UFMG) Dr. Juan Jose Garcia Pabon (coorientador Departamento de Engenharia Mecânica - UNIFEI), Dr. Ralney Nogueira de Faria (Departamento de Engenharia Mecatrônica/CEFET-MG), Dr. Tiago de Freitas Paulino (Departamento de Engenharia de Materiais/CEFET-MG), Dr. Oscar Ricardo Sandoval Rodriguez (Departamento de Engenharia Mecânica/UFMG) e Dr. Willian Moreira Duarte (Departamento de Engenharia Mecânica/UFMG), como parte dos requisitos necessários à obtenção do título de **"Doutor em Engenharia Mecânica"**, na área de concentração de **"Energia e Sustentabilidade"**.

Tese aprovada no dia 05 de setembro de 2024.

Por:

Documento assinado eletronicamente por **Luiz Machado**, **Professor do Magistério Superior**, em 13/09/2024, às 17:00, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Tiago de Freitas Paulino**, **Usuário Externo**, em 17/09/2024, às 10:20, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Juan Jose Garcia Pabon**, **Usuário Externo**, em 17/09/2024, às 10:21, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Willian Moreira Duarte**, **Professor do Magistério Superior**, em 17/09/2024, às 12:58, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Oscar Ricardo Sandoval Rodriguez**, **Membro**, em 17/09/2024, às 14:12, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Ralney Nogueira de Faria**, **Usuário Externo**, em 17/09/2024, às 18:53, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3511718** e o código CRC **26C06C42**.

Referência: Processo nº 23072.249547/2024-49 SEI nº 3511718

To all those who allowed me to envision today's achievement.

ACKNOWLEDGMENT

I thank Jesus Christ for His friendship and protection throughout my life.

I thank my parents—Marcelo de Souza Mendes and Iva Kelly de Paoli—my brother, Ricardo Cecílio Machado Junior, and my partner, Nayara Santos do Valle, for the unwavering family support provided over the years.

I thank the Federal University of Minas Gerais for its education and for the academic, financial, and various other support throughout my life, as well as the Brazilian people, whose taxes funded this research.

I thank Professor Dr. Luiz Machado for his guidance, and Professor Dr. Juan Garcia Pábon for the support and direction provided.

I thank the Refrigeration Laboratory team and everyone who contributed directly or indirectly to the completion of this work.

I thank the CAPES and CNPq, research funding agencies of the Brazilian government, for supporting this work.

My sincere thanks

"*Whoever wants to become great, must become your servant; whoever wants to be first, must be your servant.*"

Jesus Christ

"What worries me is not the cry of the bad, but the silence of the good."

Martin Luther King

RESUMO

O objetivo deste trabalho é aplicar algoritmos de deep learning no controle adaptativo de um controlador PI, atuando na manutenção da temperatura por meio de um sistema de aquecimento, ventilação e ar condicionado (HVAC) em zonas térmicas genéricas. Para isso, foram experimentados cinco algoritmos, todos de aprendizado profundo. Um sem recorrência (Deep Feedforward), um com recorrência e sem bidirecionalidade (Deep Recurrent), um com recorrência e com bidirecionalidade (Deep Bid Recurrent), um LSTM sem bidirecionalidade e um LSTM com bidirecionalidade (LSTM Bid). A necessidade de estudar a aplicação desses algoritmos é justificada por sua maior capacidade de processamento e pela falta de estudos na literatura técnica sobre seu uso na sintonia do controlador PI. Quarenta simulações de controle foram realizadas com base em dados experimentais de temperatura publicados na literatura, e em todas foi observada a estabilidade do controle no setpoint de interesse. Quanto à implementação experimental, os parâmetros de ganho estático, constante de tempo e atraso de transporte foram determinados experimentalmente para duas plantas. Uma planta foi associada a uma carga térmica de 40 W e a outra a uma carga térmica de 100 W. Com base nesses parâmetros, os ganhos do controlador foram calculados e as redes aprenderam a identificalos. Nesse contexto, foi desenvolvido um projeto de infraestrutura computacional e em nuvem para viabilizar o uso de algoritmos de deep learning no controle PI. O projeto de controle apresentado posiciona a tarefa de controle na etapa final da Indústria 4.0, tendo em vista as características preditivas dos algoritmos e a visualização do controle em tempo real, acessível a qualquer lugar do mundo com internet. Em relação aos algoritmos de deep learning utilizados para sintonia adaptativa, os algoritmos recorrentes sem bidirecionalidade mostraram-se mais vantajosos no controle adaptativo. Estes, apresentaram 48,74% menos dispersão do que o controle não adaptativo e 23,26% menos dispersão do que o controle adaptativo por redes neurais profundas sem recorrência. A bidirecionalidade não mostrou vantagens significativas para a sintonia. Por fim, o controle adaptativo por deep learning foi mais vantajoso do que o controle clássico ON/OFF, o controle não adaptativo e o ajuste direto do sinal de controle por redes neurais profundas sem integração com o algoritmo PI.

Palavras chave: Inteligência artificial; algoritmos de aprendizagem profunda; controle de temperatura; HVAC

ABSTRACT

The objective of this work is to apply deep learning algorithms to the adaptive control of a PI controller, acting in the maintenance of temperature through a heating, ventilation, and air conditioning (HVAC) system in generic thermal zones. For this purpose, five algorithms were tested, all based on deep learning. One without recurrence (Feedforward), one with recurrence and without bidirectionality (SimpleRNN), one with recurrence and bidirectionality (SimpleRNNBI), one LSTM without bidirectionality, and one LSTM with bidirectionality (LSTMBI). The need to study the application of these algorithms is justified by their greater processing capacity and the lack of studies in the technical literature on their use in PI controller tuning. Forty control simulations were performed based on experimental temperature data published in the literature, and in all cases, control stability at the desired setpoint was observed. Regarding the experimental implementation, the static gain parameters, time constant, and transport delay were experimentally determined for two plants. One plant was associated with a thermal load of 40 W and the other with a thermal load of 100 W. Based on these parameters, the controller gains were calculated, and the networks learned to identify them. In this context, a computational and cloud infrastructure project was developed to enable the use of deep learning algorithms in PI control. The presented control project positions the control task in the final stage of Industry 4.0, considering the predictive characteristics of the algorithms and real-time control visualization accessible from anywhere in the world with an internet connection. Regarding the deep learning algorithms used for adaptive tuning, the recurrent algorithms without bidirectionality proved to be more advantageous in adaptive control. These algorithms showed 48.74% less dispersion than the non-adaptive control and 23.26% less dispersion than adaptive control using deep neural networks without recurrence. Bidirectionality did not show significant advantages for tuning. Finally, adaptive control using deep learning proved to be more advantageous than classic ON/OFF control, non-adaptive control, and direct control signal adjustment by deep neural networks without integration with the PI algorithm.

Keywords: Artificial intelligence; deep learning algorithms; temperature control; HVAC

LIST OF ILLUSTRATIONS

FIGURE 1 - STAGES IN THE INDUSTRY 4.0 DEVELOPMENT PATH 16
FIGURE 2 - CONCEPTUAL HIERARCHY 19
FIGURE 3 - STRUCTURE OF THE BIOLOGICAL NEURON 20
FIGURE 4 - EXAMPLE OF ARTIFICIAL NEURON 20
FIGURE 5 - DEEP FEEDFORWARD ARCHITECTURE 21
FIGURE 6 - UNFOLDING OF THE RECURRENT NEURON 25
FIGURE 7 - LSTM CELL 29
FIGURE 8 - NEURAL NETWORK LSTM. 31
FIGURE 9 - RESULTING CLUSTER SURFACES ON THE SELF-ORGANIZING MAP (SOM) 33
FIGURE 10 - DYNAMIC NEURON 33
FIGURE 11 - HVAC PLANT CONTROLLED WITH ANN-PID CONTRAPTION 34
FIGURE 12 - EVAPORATOR UNIT AND ANN SCHEMATIC DEPICTION 34
FIGURE 13 - PROPOSED ARTIFICIAL NEURAL NETWORK TO TUNE A PID CONTROLLED 35
FIGURE 14 - CONTROL STRATEGIES 36
FIGURE 15 - SCHEMATIC VIEW OF THE NEURO-PREDICTIVE AUTOMOTIVE HVAC 37
FIGURE 16 - NEURO PID (LEFT) AND ARCHITECTURE (RIGHT) 38
FIGURE 17 - PROPOSED ANN ARCHITECTURES 38
FIGURE 18 - RECURRENT NEURAL NETWORK 39
FIGURE 19 - REINFORCEMENT LEARNING 41
FIGURE 20 - THERMAL LOAD MODEL PROPOSED AND INTERACTION WITH THE REINFORCEMENT LEARNING ANN 42
FIGURE 21 - REINFORCEMENT LEARNING APPLIED TO RESIDENTIAL HVAC CONTROL SYSTEMS 43
FIGURE 22 - MACHINE LEARNING APPROACH IN HVAC CONTROL 52
FIGURE 23 - STATE OF THE ART: PERCENTAGES. 53
FIGURE 24 - FLOWCHART METHODOLOGY FOR STEPS DEVELOPED IN THIS THESIS 56
FIGURE 25 - PLANTS IN SIMULATION CASE 57
FIGURE 26 – EXPERIMENTL SETUP AND STEP RESPONSE GRAPH FOR PLANT IDENTIFICATION 58
FIGURE 27 - PID TUNER MATLAB 60
FIGURE 28 - TENSOR DATA RECURRENT NEURAL NETWORK 62
FIGURE 29 - DASHBOARD FOR MONITORING MODELS WITH MLFLOW EXPERIMENT 62
FIGURE 30 - NEURO PI CONTROL 64

FIGURE 31 - PI CONTROL SIMULINK BLOCKS 65
FIGURE 32 - MATLAB AND PYTHON RESULTS 66
FIGURE 33 - HVAC THERMAL CONTROL. SOURCE: AUTHOR 66
FIGURE 34 - EXPERIMENTAL SETUP 70
FIGURE 35 - FREQUENCY INVERSOR AND AMPOP 71
FIGURE 36 - DATASET MODEL 72
FIGURE 37 - DASHBOARD FRONTEND SOFTWARE 73
FIGURE 38 - SOFTWARE STRUCTURE 74
FIGURE 39 - EXAMPLE OF A PYTHON ALGORITHM THAT CONSUMES AN ML MODEL 75
FIGURE 40 - SOFTWARE IN EC2 INSTANCE 76
FIGURE 41 - LABVIEW AQUISITION DATA 78
FIGURE 42 - LABVIEW GRAPHICAL INTERFACE 79
FIGURE 43 - PROJECT AWS CLOUD 80
FIGURE 44 - COMPLETE PROJECT MACHINE LEARNING CONTROL 81
FIGURE 45 - ACCUMULATION OF EPOCHS 83
FIGURE 46 - SIMULATION RESULTS BETWEEN 2990 AND 3020 SECONDS 84
FIGURE 47 - CONTROL SIMULATION FOR PLANT 6. A) 0 - 8000S, B) 2990 - 3010 S, C)5990 - 6010 S 85
FIGURE 48 - PLANT WITHOUT ADAPTIVE TUNING 86
FIGURE 49 - VARIATION OF PROPORTIONAL GAIN FOR THE FIRST PARAMETRIC CHANGE EVENT (2990 TO 3010 S) OF PLANT 6 87
FIGURE 50 - VARIATION OF THE INTEGRATIVE GAIN FOR THE FIRST PARAMETRIC CHANGE EVENT (2990 TO 3010 S) OF PLANT 6 87
FIGURE 51- EXPERIMENTAL TEMPERATURE MEASUREMENT 88
FIGURE 52 - CONTROL ON/OFF RESULTS 89
FIGURE 53 - EXPERIMENTAL NEURO-PI RESULTS 90
FIGURE 54 - RESULTS IN RANGE 450 - 650 S 91
FIGURE 55 - INTEGRATIVE PART - SEPOINT 24 92
FIGURE 56 - STANDARD DEVIATION FOR THE EXPERIMENTAL AND SIMULATION RESULTS BETWEEN THE SECONDS FOR THE PLANTS CHANGE 93
FIGURE 57 - SUM OF STANDARD DEVIATION BY MODEL 94
FIGURE 58 - DIRECTLY CONTROL DEEP LEARNING 95

LIST OF TABLES

TABLE 1 - ARTIFICIAL INTELLIGENCE ALGORITHMS 22
TABLE 2 - SUMMARY OF ANN APPLIED TO CONVENTIONAL HVAC CONTROL SYSTEM IN LITERATURE 46
TABLE 3 - MSE EXPERIMENTAL MODELS 63
TABLE 4 - PLANTS AND CONTROLLER GAINS 67

NOMENCLATURE

Greek

ω	Frequency
θ	Delay time
τ	Time constant
σ	Sigmoid function
$\emptyset$	Neural Network Bias

Latin

K	Static Gain
Q	Load Thermal
t	Time
T	Temperature
u	Controller signal
D_e	Imaginary part denominator transfer function
D_0	Imaginary part numerator transfer function
N_e	Real part denominator transfer function
N_0	Real part numerator transfer function
k	Controller Gain
U	Time step weight vector
W	Temporal weight vector
V	Output weight vector

Acronyms

LSTM	Long Short-Term Memory
MSE	Mean Squared Error
RNN	Recurrent Neural Network
SimpleRNN	Recurrent algorithm without bidirectionality
SimpleRNNBI	Recurrent algorithm with bidirectionality
LSTM	LSTM algorithm without bidirectionality
LSTMBI	LSTM algorithm with bidirectionality
FEEDFORWARD	Algorithm without recurrence

Subscripts

p	Proportional
i	Integrative
d	dreivative

SUMMARY

1 INTRODUCTION.....15

1.1 Research relevance and innovations.....16
1.2 General objectives.....17
1.3 Specific objectives.....17

2 LITERATURE REVIEW.....18

2.1 Artificial intelligence.....19
2.2 Recurrent Neural Network.....24
2.3 Backpropagation in time.....25
2.4 Recurrent Neural Network LSTM.....29
2.5 Applications of artificial intelligence in HVAC control.....32
2.6 Closure for literature review.....50

3 METHODOLOGY.....54

3.1 Plant identification 57
3.2 Stability analysis for multiple plants 59
3.3 Training of neural networks 60
3.4 Neuro-PI Control 63
3.5 Integration of neural networks into simulation 64
3.6 Experimental setup 68
3.7 Project software machine learning control 72
3.8 Closure for methodology.....82

4 RESULTS AND ANALYSIS.....83

5 CONCLUSIONS.....96

6 SUGGESTIONS FOR FUTURE WORK.....98

7 ACKNOWLEDGMENT.....100

BIBLIOGRAPHY.....101

APPENDIX.....110

1 Dense Neural Network.....110
2 Recurrent Neural Network.....112
3 Recurrent LSTM Neural Network.....115
4 Bidirectional LSTM.....117
5 Machine Learning experiment in MLFLOW inside Google Collab..... 120
6 Python environment simulation.....124
7 Raspberry pi code.....128
8 Commands Docker, Terraform e AWS CLI.....130

1 INTRODUCTION

Due to the significant increase in global energy consumption in recent years, developing and implementing more efficient control operations and equipment has become crucial. Specifically for heating, ventilation and air conditioning (HVAC) system. In buildings, these systems accounted for 30% of global energy consumption in 2019 (UNEP). As the world rapidly approaches a scenario where emissions need to be reduced without compromising economic viability, developing and implementing effective control techniques for better energy utilization becomes imperative.

Temperature control through HVAC systems is essential for many processes. Various methods are used for this, including Classical Control, Hard Control, Soft Control and Hybrid Control (Afram and Janabi-Sharifi, 2014). Classical Control techniques involve On/Off and P, PI, and PID algorithms. Hard Control focuses on nonlinear systems requiring advanced mathematical development. Soft Control applies Fuzzy logic and artificial neural networks, while Hybrid Control combines classical and soft control methods, such as using artificial neural networks for PID controller tuning.

In Hybrid Control, simple feedforward neural networks are commonly used for adaptive tuning of PI/PID controllers (Afram and Janabi-Sharifi, 2014). However, there is a lack of research using deep recurrent neural networks or deep feedforward networks for this purpose. For instance, was not found works that explore the use of Long Short-Term Memory (LSTM) recurrent neural networks in tuning PI/PID controllers. A systematic review by Halhoul et al. (2021) did not identify any work employing deep recurrent neural networks for this task. Similarly, D. Lee and Lee (2023) reviewed 1700 studies from 88 authors, with none investigating adaptive tuning of PI/PID controllers using LSTM networks.

Nevertheless, some studies have explored simple recurrent neural networks for tuning, including work by Varshney and Sheel (2011), Chiang et al. (2014), Reichensdörfer et al. (2017), and Bouzaiene et al. (2021). However, these studies did not consider the effects of using short and long-term recurrent neural networks (LSTM), impact of bidirectionality on the tuning process and the models are simple and not deep layers.

The application of deep recurrent neural networks and deep feedforward neural networks in PI controller tuning remains an underexplored area compared to simple feedforward networks. To support this claim, a search using the Research Rabbit tool (2021) revealed 733 related studies from 1985 to 2024, none of which examined the feasibility of using deep recurrent neural networks, with or without bidirectionality, or LSTM networks in PID/PI controller tuning for

HVAC temperature control. However, with the increase in sensor data in plants and the consequent increase in data volume, artificial neural networks with greater processing capacity would become increasingly necessary.

In relation to the current technological moment, Figure 1 was presented in the work of Schuh et al. (2020) and shows the maturity levels for an industry to be classified as part of the fourth industrial revolution. The third item, in Industry 4.0, should have algorithms with strong predictive capabilities, enabling them to predict what will happen based on the data. The final step for a company to be considered part of Industry 4.0 is to be able to produce an automated response based on the predictive capabilities of the previous step.

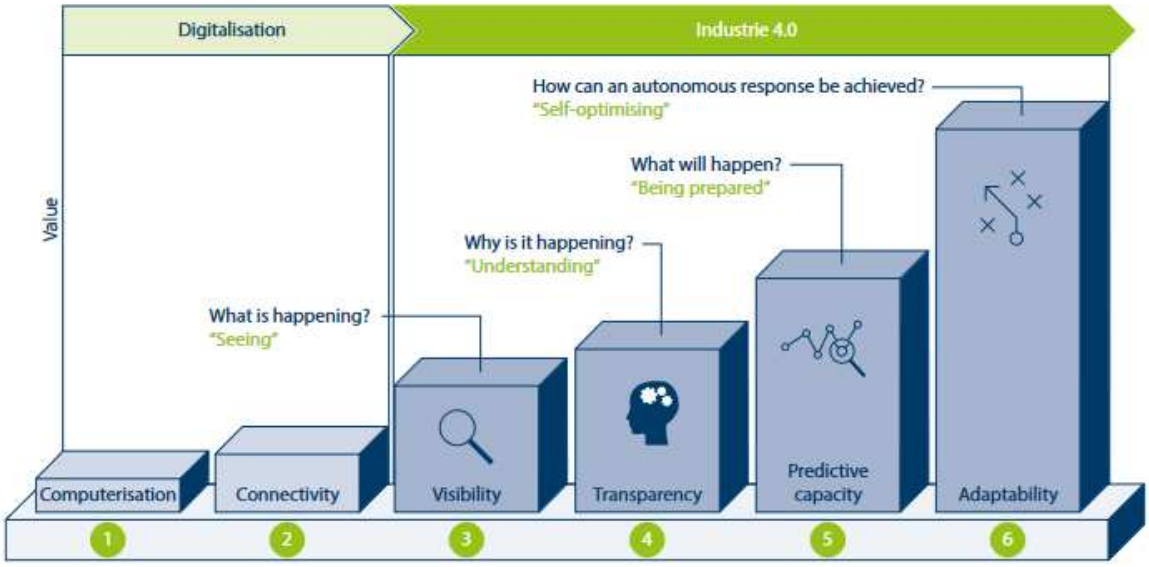


Figure 1 - Stages in the Industry 4.0 development path (Schuh et al. 2020)

Considering item 5 in Figure 1, deep feedforward neural networks, deep recurrent networks, and LSTMs have greater predictive capabilities compared to simple neural networks and are therefore more capable of operating with large volumes of data. It is, accordingly, worthwhile to study their behavior in the adaptive tuning of the PI controller. As for item 6, concerning how to implement these algorithms to generate automated responses, the methodology chapter of this thesis will present a project that addresses this question in the context of the refrigeration industry, using adaptive PI control for temperature regulation.

1.1 Research relevance and innovations

There are no technical papers in the literature that utilize Deep Learning algorithms in adaptive PI control acting on ventilation, heating, and air conditioning (HVAC) systems.

However, with technological development, these types of algorithms will be necessary for the control of large-scale plants that require a significant amount of data for identification.

1.2 General objective

Control temperature using Deep Learning algorithms for the adaptive control of the Proportional-Integral (PI) controller in heating, cooling, and ventilation systems of an air conditioning (HVAC) system.

1.3 Specific objectives

- Present a decision-making map of which artificial intelligence algorithms to use based on the control objectives.
- Detail the software project that allows the use of Cloud-based Deep Learning algorithms and position the temperature control at the latest stage of Industry 4.0.
- Study the PI control adaptively tuned by recurrent Deep Learning algorithms without bidirectionality and with bidirectionality.
- Study the PI control adaptively tuned by LSTM (Long Short-Term Memory) Deep Learning algorithms with and without bidirectionality.
- Compare the results of adaptive control by Deep Learning with non-adaptive control, ON/OFF control, and Deep Learning without using the PI controller.

2. LITERATURE REVIEW

Artificial Neural Networks (ANNs) can learn through supervised, unsupervised, and reinforcement processes. In supervised processes, there are input and output data through which the algorithm can learn the relationship between them. There are various algorithm configurations that make use of the supervised training process, used as instant data detection models such as Multilayer Perceptron's (Ganowicz et al. 2022), as predictive models like Recurrent Neural Networks (Mao and Sejdic 2022), or as image identification models like Convolutional Neural Networks (Zafar et al. 2022).

In unsupervised training (Xiao et al. 2023) , there are typically input data but no output data. A neural network trained in this context learns the relationships between data points by grouping them into proximity regions. For example, suppose the data points represent people from different ethnic backgrounds. After unsupervised training, the neural network would identify similarities among people from Nordic countries or Asian countries. Thus, if it were used to identify the ethnicity of a Danish citizen, it would classify that person as Nordic.

Algorithms that use reinforcement learning techniques (Matsuo et al. 2022 ; (Huang et al., 2022; Jendoubi and Bouffard, 2023), are the ones that come closest to human learning. In this type of technique, the algorithm must interact with an environment and seek, through a system of rewards and penalties, the best course of action within that environment while pursuing a previously known objective. To illustrate, in human learning, you repeat a specific task until you can execute it perfectly. Before reaching the goal, you make mistakes and incur penalties for those errors. As you start to get things right, you receive positive feedback for your actions and adjust your next steps accordingly until you achieve mastery of the task and ultimately reach the objective.

The training methodologies mentioned for artificial neural networks allow the training of algorithms that can be used either independently or in conjunction with control algorithms to act on mechanical components of the HVAC system. This includes, for example, adjusting the mass flow rate of the refrigerant passing through an electronic expansion valve, controlling the fan speed for air inflow in both the evaporator and condenser, as well as regulating compressor speed and various other elements directly or indirectly associated with the HVAC circuit. This enables fine-tuning of thermal control tasks. In item 2.1, will go more deeply into artificial intelligence. In the item 2.2, will narrow the focus to recurrent neural networks, and finally, in item 2.3, a review of the state-of-the-art applications of machine learning algorithms in HVAC system control will be presented.

2.1 Artificial intelligence

Artificial intelligence (AI) is a broader field of computer science that includes several subareas, one of which is machine learning. Artificial Intelligence refers to systems or machines that mimic human intelligence to perform tasks. It includes various technologies and methods, such as machine learning, natural language processing (NLP), expert systems, and artificial neural networks, among others. Figure. 2 illustrates the relationship between these commonly used terms.

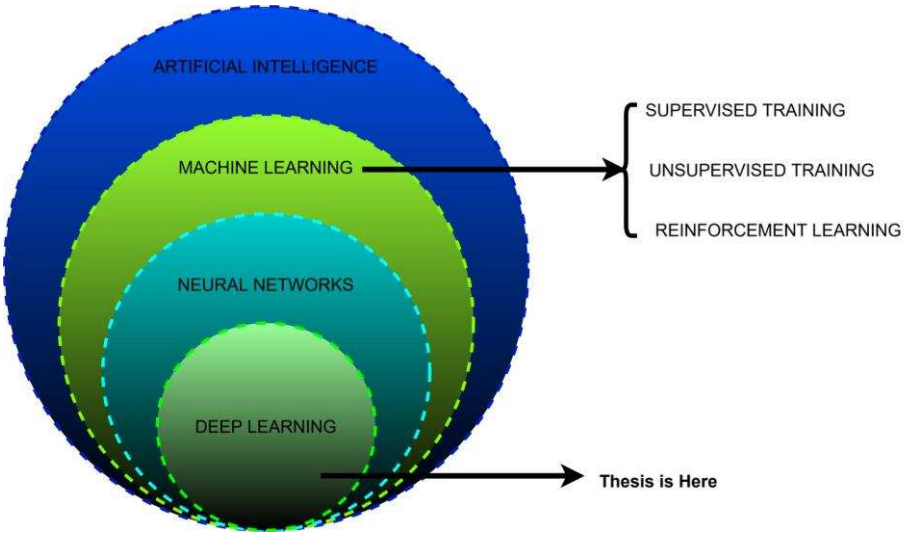


Figure 2 - Conceptual Hierarchy. Source: Author

Machine Learning is therefore a subset of AI that focuses on developing algorithms that enable computers to learn and to make predictions or decisions based on data. Instead of being explicitly programmed to perform a task, these algorithms use patterns and inferences to improve performance on specific tasks. Among the data-driven techniques are artificial neural networks, which range from neural structures with few intermediate layers and lower processing capacity to more complex structures commonly referred to as Deep Learning.

The first study related to neurocomputing was by W.S McCulloch in 1943 (Block 1962), who modeled the functioning of the biological neuron as a system composed of Boolean logic, resulting in the first conception of an artificial neuron. The fundamental cell of the central nervous system is the neuron, which functions by propagating electrical signals between its ends, establishing contact between one neuron and another. Figure 3 illustrates the neuron cell.

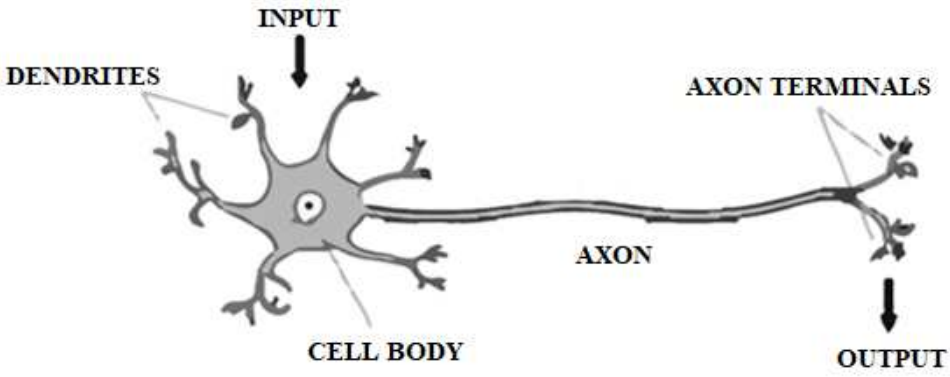


Figure 3 - Structure of the Biological Neuron. Adapted from: Neves et al. (2018)

The artificial neuron functions analogously to the biological neuron. It is the most basic cell of an artificial neural network. In the artificial neuron, the input layer functions as the dendrites, the summation of the 'signals' is computed by the activation function—similar to what happens in the nucleus of the biological cell—and, depending on the value, the output signal is emitted. The Figure. 4 represents the artificial neuron.

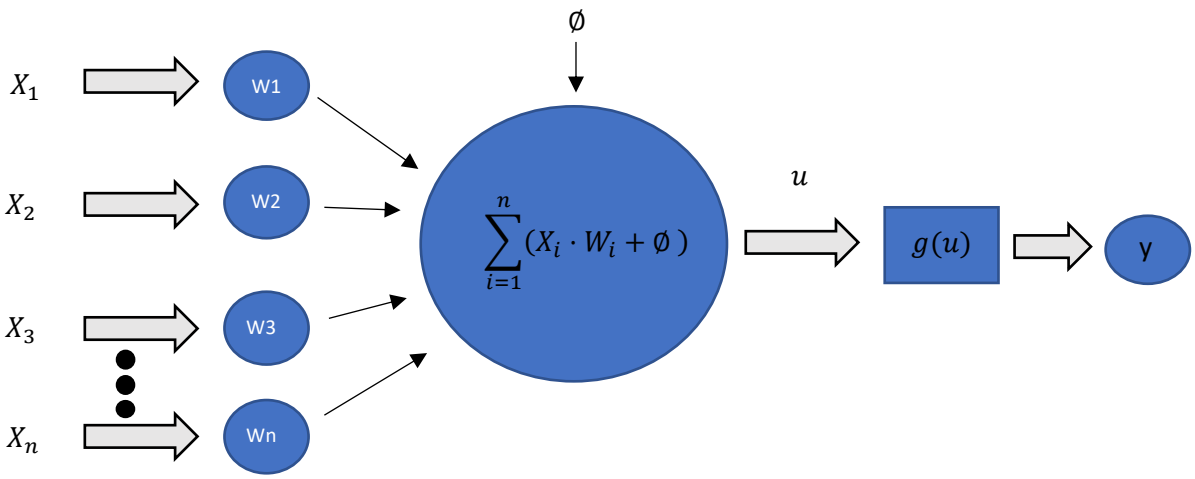


Figure 4 - Example of Artificial Neuron. Source: Adapted of KeLin Du et al. (2022)

The input signals described by $(X_1, X_2 \ldots X_n)$ are the inputs, and the synaptic weights $(w_1, w_2 \ldots w_n)$ quantify the relevance of the signal for that specific process. The input signals, weighted by the synaptic weights, are summed and added to the activation threshold (θ). This value, when added to the weighted sum, produces what is call the activation potential (u). If the activation potential is positive, the neuron will produce an excitatory signal; if it is negative, it will produce an inhibitory signal. The activation potential is used as input to the activation function (g(u)), which determines the neuron's output (y) within a set of reasonable values.

The architecture of an artificial neural network refers to how its neurons are interconnected. It is generally described by the direction of the propagated signal. Essentially, the architecture of traditional neural networks can be divided into three parts: the input layer, the hidden layers, and the output layer. The input layer is responsible for receiving the input data to the network, which are usually normalized according to the dynamic range of the activation functions. The hidden layers map the relationship between the input and output signals, performing the processing that allows the network to learn. Finally, the output layer presents the final results.

Figure 5 shows an example of a deep FeedForward architecture, consisting of an input layer with eight neurons, three hidden layers—the first with eleven neurons and one bias, the second with five neurons and one bias, and the third also with eleven neurons and one bias. The output layer is the last layer and contains two neurons. The number of hidden layers can vary; if there is more than one, the architecture becomes a deep neural network.

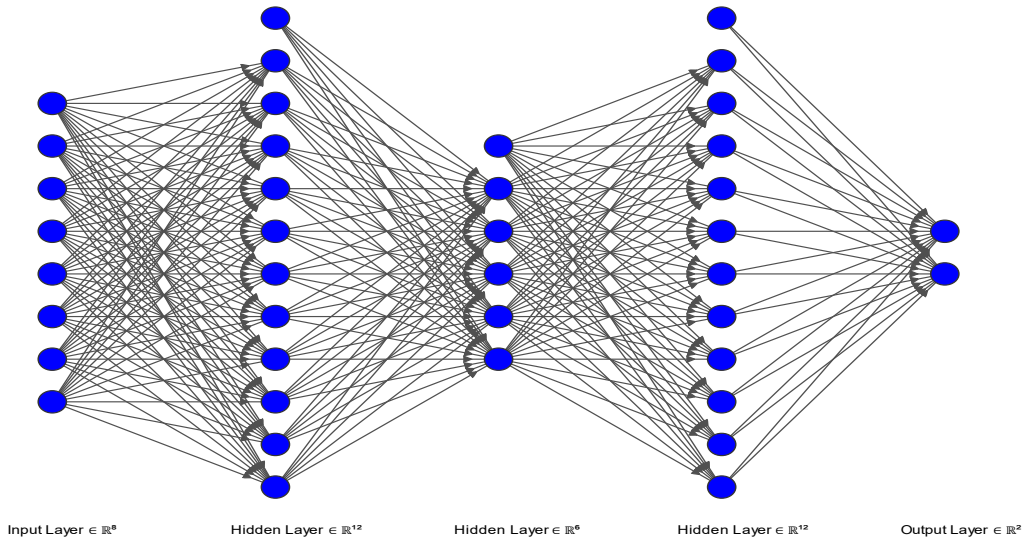


Figure 5 - Deep Feedforward Architecture. Source: Author

The training of the deep Feedforward artificial neural network presented in Figure 5 occurs through regression on a dataset between its input and output. There are various types of algorithms that use regression to generate learning, such as: Locally Weighted Regression (LWR) (Atkeson and Schaal 1995), XCR for Function (XCSF) (Butz and Herbort 2008), Model Trees (MS) (Kotsiantis 2013), Kernel Ridge Regression (KRR), among others. However, the learning of artificial neural networks typically occurs using gradient descent (Ruder 2016) combined with backpropagation (Wythoff 1993) across multiple neural layers, which differs from the use of other regression-based algorithms as mentioned earlier. Table 1 presents some

of the most commonly used types of algorithms of artificial intelligence and their descriptions. Those used in the research for this thesis are highlighted in yellow in the table.

Table 1 - Artificial intelligence algorithms

Name	Description	Reference
Naive Bayes	Probabilistic classifier based on Bayes' Theorem.	(Frank et al. 2000)
Logistic Regression	Used for binary or multiclass classification.	(LaValley 2008)
K-Means Clustering	Partitions the data into K distinct clusters.	(Cheung 2003)
Suport Vector Machines	Used for classification and regression, based on maximizing the margin between classes.	(Shmilovici 2010)
K-Nearest Neighbors	Classification and regression based on proximity to nearest neighbors.	(S. Sun and Huang 2010)
Decision Trees	Rule-based models for classification and regression.	(de Ville 2013)
DBSCAN	Methods that directly optimize the policy (rather than learning a value function), adjusting the policy parameters in the direction of the gradient of the expected reward.	(Fong et al. 2014)
Policy Gradient	Methods that directly optimize the policy (rather than learning a value function), adjusting the policy parameters in the direction of the gradient of the expected reward.	(Silver et al. 2014)
Spectral Clustering	Uses graph theory to find data structure.	(Jia et al. 2014)
Hierarchical Clustering	Creates a hierarchy of clusters.	(Bouguettaya et al. 2015)
Deep Q-Network	Extension of Q-Learning that uses deep neural networks to approximate the Q-value function.	(Arulkumaran et al. 2017)
Stocastic Gradient Descendent	Creates a hierarchy of clusters.	(Ketkar 2017)
Proximal Policy Optimization (PPO)	Algorithm that uses a form of policy optimization that is stable and efficient, adjusting the policy within a trustworthy region to ensure stable updates.	(Schulman et al. 2017)
SARSA	On-policy algorithm that updates the Q-value function using the action that the current policy will choose in the next step.	(Zhao et al. 2017)
(TRPO)	Algorithm that adjusts the policy within a "trust region" to ensure safe and stable updates.	(Aubert 2017)

Continue...

Expectation Maximization	Iterative technique for finding parameters of statistical models.	(Van Steenkiste et al. 2018)
Bagging	Ensemble technique that combines predictions from multiple models.	(Pham, Tien Bui, and Prakash 2018)
Mini Batch K-Means	Neural network architectures designed to handle sequential data, using attention mechanisms instead of recurrence to capture long-range dependencies.	(Peng, Leung, and Huang 2018)
Convolutional Neural Networks (CNN)	Neural networks designed to process data with a regular grid, such as images. They use convolutional layers to capture spatial features and local patterns.	(Voulodimos et al. 2018)
Actor-Critic Methods	Combination of policy-based (actor) and value-based (critic) methods, where the actor updates the policy and the critic estimates the value function.	(Haarnoja et al. 2018)
AdaBoost	Ensemble method that weights weak classifiers.	(F. Wang et al. 2019)
Random Forest	Ensemble of decision trees to improve accuracy.	(Parmar, Katariya, and Patel 2019)
MobileNet	Efficient and lightweight neural networks designed for mobile devices and embedded systems, using depthwise separable convolutions.	(Sinha and El-Sharkawy 2019)
Long Short-Term Memory (LSTM)	Type of RNN capable of learning long-term dependencies, overcoming the vanishing gradient problem.	(Smagulova and James 2019)
Q-Learning	Off-policy reinforcement learning algorithm that seeks to find the optimal policy by learning a Q-value for each state-action pair.	(Clifton and Laber 2020)
Linear Regression	Used to predict continuous values.	Kumar and Bhatnagar 2021)
Gradient Boosting Machines	Ensemble technique that creates a strong model from weak models.	(Bentéjac, Csörgő, and Martínez-Muñoz 2021)
Recurrent Neural Networks (RNNs)	Neural networks designed to process sequential data, where the connections between nodes form a directed graph along a temporal or sequential sequence.	(J. F. Torres et al. 2021)
Autoencoders	Neural networks used to learn efficient representations (encodings) of data, typically for dimensionality reduction or noise removal.	(Baur et al. 2021)
Deep Belief Networks (DBNs)	Deep neural networks consisting of multiple layers of restricted Boltzmann machines (RBMs), where each	(Sohn 2021)

Continue...

	layer learns from the previous layer in an unsupervised manner.	
OPTICS	Similar to DBSCAN, but handles clusters of varying density better.	(Y. Guo et al. 2021)
Generative Adversarial Networks (GANS)	Neural networks consisting of two parts, a generator and a discriminator, that compete with each other. The generator creates fake samples, while the discriminator tries to distinguish between real and fake samples.	(Aggarwal, Mittal, and Battineni 2021)
DenseNet	Neural networks where each layer is connected to all previous layers, facilitating the flow of information and gradients.	(Ke-Lin Du, Chi-Sing, Wai Ho 2022)
EfficientNet	Neural network architectures that simultaneously optimize depth, width, and resolution of the network to improve efficiency.	(Nayak et al. 2022)
Principal Component Analysis (PCA)	Transforms data into new orthogonal variables that capture most of the variance in the data.	(C. Liu et al. 2022)
Monte Carlo Tree Search	Algorithm that iteratively builds a search tree and uses Monte Carlo simulations to determine the best moves or actions.	(Świechowski et al. 2023)
Transformers Networks	Neural network architectures designed to handle sequential data, using attention mechanisms instead of recurrence to capture long-range dependencies.	(Maurício, Domingues, and Bernardino 2023)
YOLO (You Only Look Once)	Algorithm used for image recognition and detection.	(Zhou 2024)

2.2 Recurrent neural network

Recurrent neural networks differ from feedforward neural networks due to the feedback within the neurons. This characteristic allows for an understanding of the context in which the data is embedded, based on a temporal or sequential order. It is widely used in natural language processing (S. Liu et al., 2014) because words in a text are fully understood according to the context in which they are used. While feedforward neural networks (Ramchoun et al. 2016) consider input and output data in isolation, recurrent neural networks take into account both the current data and data from previous moments. Figure 6 shows the unfolding of a recurrent neuron over time in three-time steps, as well as the structure of a recurrent neural network with an input layer with four inputs, two hidden layers with five neurons in the first and four neurons in the second, and an output layer with two recurrent neurons.

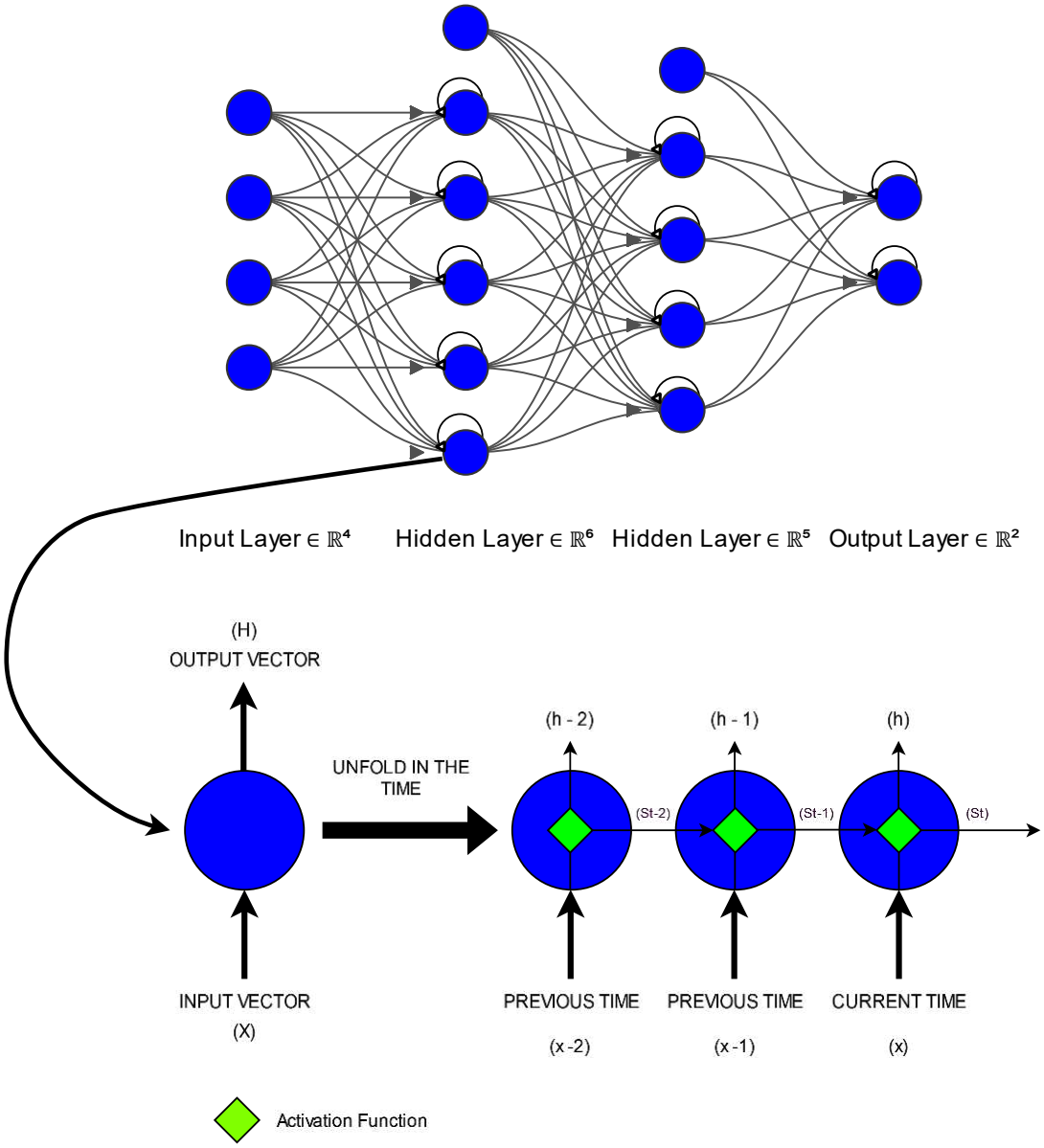


Figure 6 - Unfolding of the Recurrent Neuron. Source: Author

2.2.1 Backpropagation in time

As presented in Figure 6, for an algorithm that seeks to understand the relationship between a present time step (x) and two previous time steps $x_{(t-1)}$ and $x_{(t-2)}$, the hidden state (s) at each time step is processed considering the weight vectors (U) and (W), where U is the weight vector for the inputs at the current time step and W is the weight vector for the temporal dependencies, as well as the state from the previous time step $\left(s_{(t-n)}\right)$ and the activation threshold for the current state (b_t). Equation 1 calculates the internal state at the first-time step.

$$s_{(t-2)} = \sigma\left(U \cdot x_{(t-2)} + W \cdot 0 + b_{(t-2)}\right) \quad 1$$

The first hidden state $s_{(t-2)}$ is usually initialized with the temporal weight vector (W) multiplied by zero since there is no previous time step for this example. All time steps are weighted by the weight vector (U). The subsequent hidden state ($s_{(t-1)}$) is calculated considering the previous hidden state (s_{t-2}), , as shown in Equation. 2.

$$s_{(t-1)} = \sigma\left(U \cdot x_{(t-1)} + W \cdot s_{(t-2)} + b_{(t-1)}\right) \quad 2$$

Where (σ) is the sigmoid function. The current hidden state (s_t) is calculated considering the previous hidden state (s_{t-1}), as shown in Equation 3.

$$s_{(t)} = \sigma\left(U \cdot x_{(t)} + W \cdot s_{(t-1)} + b_{(t)}\right) \quad 3$$

The interconnection between hidden states creates the memory effect of the recurrent neural network. In addition to the hidden states, an output (h) is calculated at each time step, which will later be used for backpropagation through time during the training process. These outputs are calculated using the softmax activation function and weighted by the weight vector (V). Equations 4, 5, and 6 calculate the outputs for the time steps at ($x_{(t-2)}, x_{(t-1)}\ and\ x_{(t)}$).

$$h_{(t-2)} = \text{softmax}\left(V \cdot s_{(t-2)} + b_{(t-2)}\right) \quad 4$$

$$h_{(t-1)} = \text{softmax}\left(V \cdot s_{(t-1)} + b_{(t-1)}\right) \quad 5$$

$$h_{(t)} = \text{softmax}\left(V \cdot s_{(t)} + b_{(t)}\right) \quad 6$$

The learning of recurrent neural networks uses backpropagation through time (Lillicrap and Santoro 2019). Unlike the backpropagation in feedforward networks (HECHT-NIELSEN, 1992) , it considers the gradients at all time steps of the dataset. Consider the unrolled RNN neuron in Figure 6. Each time step has a hidden state $\left(s_{(t-n)}\right)$ at time (t_n) and receives an input($x_{(t-n)}$) at the same time. The cell's output is denoted by $h_{(t-n)}$. The objective is to minimize a total loss function (L), which is the sum of the three loss functions, calculated by comparing the RNN's output $\left(h_{(t-n)}\right)$ with the expected output $\left(h^*_{(t-n)}\right)$. To achieve this, it is necessary to calculate the local gradients of the weight vectors U, V, and W so that these vectors

can be updated during the learning process using the gradient descent algorithm (Ruder 2016). Equations 7, 8, and 9 present the updates for the vectors V, W, and U, which are the same for all time steps, considering the gradient (δ) and the learning rate (η).

$$V_i = V_{(i-1)} - \eta \cdot \frac{\delta L}{\delta V} \quad (7)$$

$$W_i = W_{(i-1)} - \eta \cdot \frac{\delta L}{\delta W} \quad (8)$$

$$U_i = U_{(i-1)} - \eta \cdot \frac{\delta L}{\delta U} \quad (9)$$

For each time step, is calculate a loss (L_n), which can be of different types such as mean square error (MSE) or cross-entropy. The total loss is the sum of the losses over the three-time steps. Therefore, the total gradient is the sum of the gradients, as shown in Equations 10 - 12.

$$\frac{\delta L}{\delta V} = \frac{\delta L_1}{\delta V} + \frac{\delta L_2}{\delta V} + \frac{\delta L_3}{\delta V} = \sum_{1}^{3} \left(\frac{\delta L_n}{\delta V}\right) \quad (10)$$

$$\frac{\delta L}{\delta W} = \frac{\delta L_1}{\delta W} + \frac{\delta L_2}{\delta W} + \frac{\delta L_3}{\delta W} = \sum_{1}^{3} \left(\frac{\delta L_n}{\delta W}\right) \quad (11)$$

$$\frac{\delta L}{\delta U} = \frac{\delta L_1}{\delta U} + \frac{\delta L_2}{\delta U} + \frac{\delta L_3}{\delta U} = \sum_{1}^{3} \left(\frac{\delta L_n}{\delta U}\right) \quad (12)$$

The individual gradients (δL_t) for each time step can be calculated using the chain rule. For the gradients of the cost function with respect to the output vector (V) and considering $o_t = V \cdot s_{(t)} + b_{(t)}$, it is possible obtain equation 13.

$$\frac{\delta L_t}{\delta V} = \frac{\delta L_t}{\delta h_t} \cdot \frac{\delta h_t}{\delta o_t} \cdot \frac{\delta o_t}{\delta V} \quad (13)$$

The calculation of the gradient of the cost function with respect to the output vector V is relatively simpler than with respect to the vectors W and U. This is because the vectors W and U have interdependence between the time steps, as they are involved in the calculation of the internal state at all three-time steps, as seen in Equations 1, 2, and 3. The gradients with respect to W and U can be calculated using the chain rule, as shown in Equations 14 and 15.

$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta h_t} \cdot \frac{\delta h_t}{\delta o_t} \cdot \frac{\delta o_t}{\delta h_t} \cdot \frac{\delta h_t}{\delta W} \quad 14$$

$$\frac{\delta L_t}{\delta U} = \frac{\delta L_t}{\delta h_t} \cdot \frac{\delta h_t}{\delta o_t} \cdot \frac{\delta o_t}{\delta h_t} \cdot \frac{\delta h_t}{\delta U} \quad 15$$

However, to calculate the local gradient with respect to the weight vector W and the vector U, it is necessary, according to the chain rule, to calculate the derivative of the hidden state (s_t) with respect to W and U, taking into account both the direct gradient considering the current moment and the indirect gradients from past moments. Since the hidden state s_t considers $s_{(t-1)}$, which in turn considers $s_{(t-2)}$, this dependency would continue for more time steps up to $s_{(t-\infty)}$.

$$\frac{\delta h_t}{\delta W} = \frac{\delta h_t}{\delta W} + \frac{\delta h_t}{\delta h_{t-1}} \cdot \frac{\delta h_{t-1}}{\delta W} \quad 16$$

$$\frac{\delta h_t}{\delta U} = \frac{\delta h_t}{\delta U} + \frac{\delta h_t}{\delta h_{t-1}} \cdot \frac{\delta h_{t-1}}{\delta U} \quad 17$$

Equations 16 and 17 demonstrate the interconnection between the states at the current and previous times. However, there is also the same relationship between the internal state h_{t-1} and h_{t-2}, which makes them equation 18 and 19.

$$\frac{\delta h_t}{\delta W} = \frac{\delta h_t}{\delta W} + \frac{\delta h_t}{\delta h_{t-1}} \cdot \left(\frac{\delta h_{t-1}}{\delta W} + \frac{\delta h_{t-1}}{\delta h_{t-2}} \cdot \frac{\delta h_{t-2}}{\delta W} \right) \quad 18$$

$$\frac{\delta h_t}{\delta U} = \frac{\delta h_t}{\delta U} + \frac{\delta h_t}{\delta h_{t-1}} \cdot \left(\frac{\delta h_{t-1}}{\delta U} + \frac{\delta h_{t-1}}{\delta h_{t-2}} \cdot \frac{\delta h_{t-2}}{\delta U} \right) \quad 19$$

Therefore, there is a temporal/sequential dependency between the gradients of a given state and the previous states. This dependency can lead to problems such as gradient explosion or gradient vanishing (Montreal 2013) when using a training dataset with many time steps. To avoid this situation, it is common to truncate backpropagation (Tallec and Ollivier 2017) to a certain number of dependencies. Once the local gradients are obtained, the weights are updated using the gradient descent algorithm (Equations 7 – 9) until the error between the network's predicted output and the expected output is acceptable

2.2.2 Recurrent Neural Networks LSTM

While simple recurrent neural networks can understand the relationship between data with a certain level of proximity, LSTM (Long Short-Term Memory) networks are better at comprehending relationships between both closely related and more distantly spaced data. Unlike the simple neural network represented by Figure 4, the LSTM is a cell composed of four internal neural networks, known as the three gates of the LSTM. Figure 7 shows the cell unrolled over three time steps and presents the LSTM gates, which, from left to right, are the forget gate, the input gate, and the output gate.

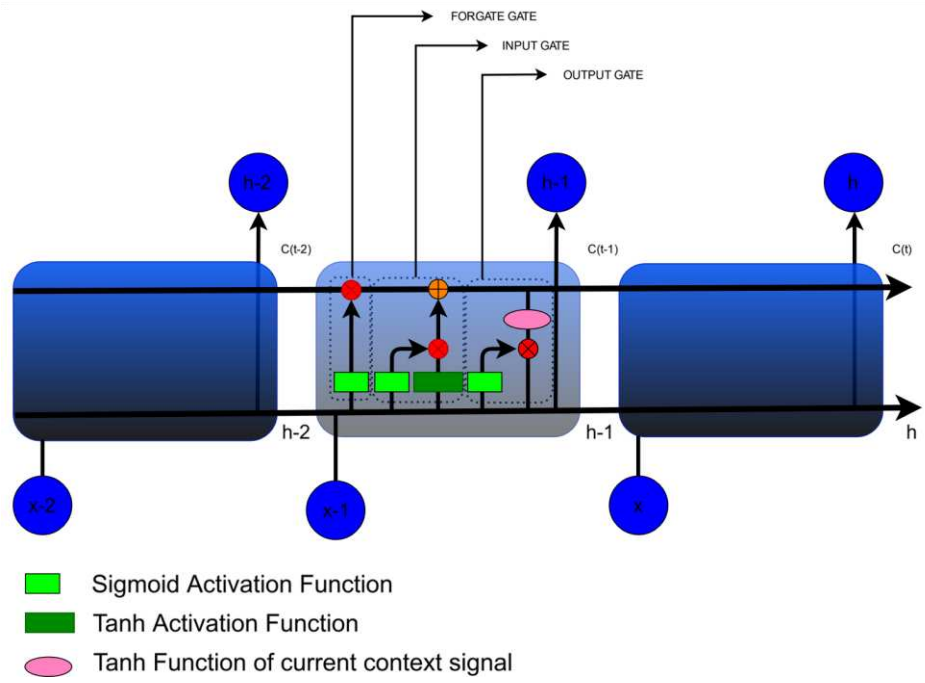


Figure 7 - LSTM Cell. Source: Author

The memory of LSTM cells is stored in the state signal (C), represented by the top horizontal line that spans across the time steps in Figure 7. At the current time step, it is

necessary to decide whether the information (h-1) is important, partially important, or not needed in the context of the data. This decision is made by the forget gate, which uses the state signal from the previous cell concatenated with the current input (x) and weighted by the weights (W_{ig}) and activation threshold (b_{ig}), through a sigmoid activation function. This transforms the signal into values between 0 and 1. If it is 0, the information is irrelevant and should be forgotten; if it is 1, it should be fully remembered. Intermediate values indicate information that should be partially remembered. This memory effect is achieved by multiplying the activation function signal of the forget gate by the previous state signal (ct-1). How presented in equation 20.

$$f_{fg} = \sigma\left(W_{fg} \cdot \left[h_{(t-1)}, x_t\right] + b_{fg}\right) \quad 20$$

The next step is to decide which new information will be stored in the current cell state (ct). This is done by the input gate. This gate consists of two neural networks. The first network, with a sigmoid activation function, determines which values should be updated. The value from this network (i_{ig}) is multiplied by the output of another network with a hyperbolic tangent activation function, which generates new candidates for the cell's context ($\tilde{C}_{ig}$). The result of this product is added to the memory state signal.

$$i_{ig} = \sigma\left(W_{ig} \cdot \left[h_{(t-1)}, x_t\right] + b_{ig}\right) \quad 21$$

$$\tilde{C}_{ig} = tanh\left(W_c \cdot \left[h_{(t-1)}, x_t\right] + b_c\right) \quad 22$$

$$C(t) = f_{fg} \cdot C(t-1) + i_{ig} \cdot \tilde{C}_{ig} \quad 23$$

After obtaining the current state signal, it will pass through the output gate to form the signal (h_t) for the next LSTM cell to analyze. The output gate acts as a filter, weighing the signal (h_{t-1})concatenated with (x_t) using a sigmoid activation function and multiplying by the hyperbolic tangent of the current context signal. The result of this operation is the output of the current cell (h_t)

$$h(t) = \left(\sigma\left(W_o \cdot \left[h_{(t-1)}, x_t\right] + b_o\right)\right) \cdot \left(\tanh\left(C(t)\right)\right) \quad 24$$

The training of the LSTM network updates the weight vectors (W) and biases (b) using backpropagation through time, as described in the previous section, and gradient descent methods until the outputs at the time steps provided by the network match the training data within an acceptable margin of error. Since the LSTM is a cell with four internal neural networks, its architecture is different from recurrent neural networks like the one represented in Figure 6. Figure 8 shows an LSTM architecture with two layers, each containing 5 neural cells, unrolled over three-time steps. The output of the last layer at the current time step is connected to a dense neural network that produces two outputs. It would also be possible to consider all time steps in the output of the last LSTM layer as input to the dense layer, but this would increase computational cost.

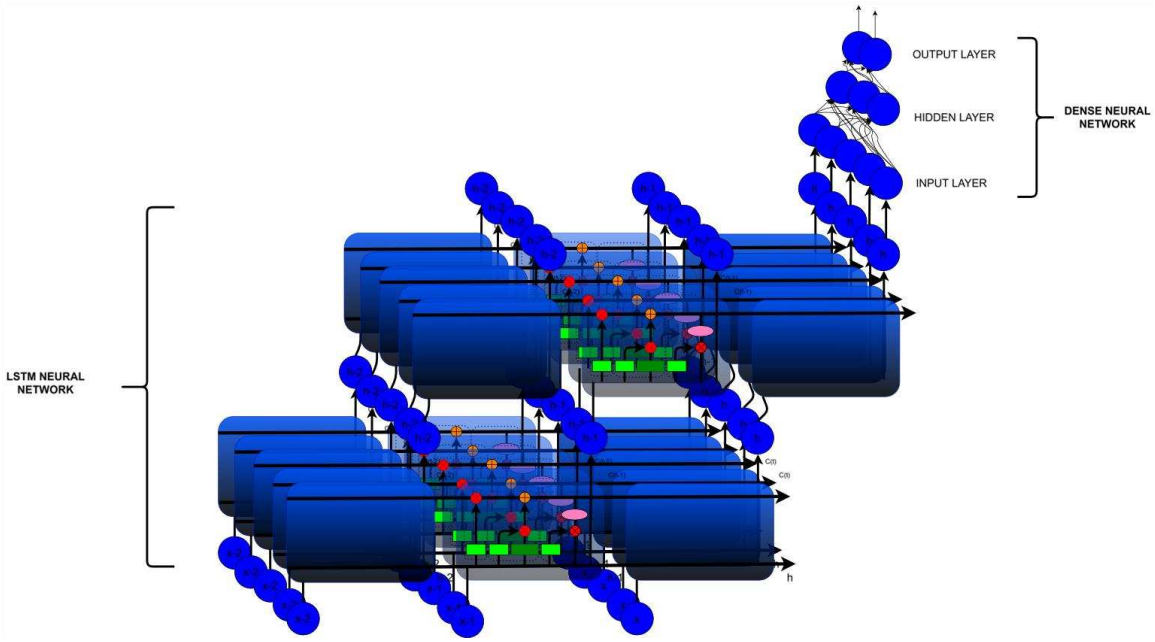


Figure 8 - Neural Network LSTM. Source: Author

In some situations, knowing the context of the data prior to the data to be predicted is sufficient. However, in many applications, it is necessary to understand not only the past context of the information but also the future context. A neural network that considers both past and future context is bidirectional. This adds an extra layer that processes both the FeedForward step and the Backward step simultaneously. Thus, it considers information both forward and backward, with the trade-off being an increase in computational cost.

2.3 Applications of artificial intelligence in HVAC control

Teeter and Chow (1998) proposed utilizing an simple recurrent neural networking (RNN) to control an HVAC system, by estimating the future heat requirements imposed by the climatized region, it would be capable of preventively adjusting the system operating parameters, such as return air mass flow rate and fan rotational speeds. The authors concluded the RNN was capable of predicting upcoming dynamic changes, and thus, achieve an efficient system operating performance.

Ueda and Taniguchi (1999) proposed a model based on ANN to predict the thermal comfort levels in a passenger car. The authors utilized the passengers facial and cabin temperatures as input data, and the ANN predicted the cabin thermal comfort levels. By proposing a perceptron ANN, with four neurons on the input and intermediate layers, and a single neuron output layer, it was possible to establish and maintain superior thermal control in the system.

Kajino et al. (2000) proposed an Artificial Neural Network architecture to control a Toyota Progress AC evaporator fan rotational velocity. In their study, the ANN controlled air conditioning unit reduced the fuel consumption whilst maintaining appropriate ventilation levels in the climatized cabin. Moreover, the authors concluded that ANN approaches could, in the future, decrease the human interference in the HVAC unit controller tuning and development.

Henze and Hindman (2002) studied a chiller condenser using its operating data as inputs to a unsupervised ANN, whose goal was the generation of a Self-Organizing Map (SOM) (Xu et al. 2023). Self-Organizing Maps are unsupervised machine learning techniques that allow higher dimensional data to be represented in lower dimensions, enabling data converging regions (referred to as clusters) to be visualized. With the Self-Organizing Map Henze and Hindman developed several different Feedforward ANN's, with radial basis activation function to control the fans frequency. Figure **9** presents the Self-Organizing Map used to help make control decisions depending on the region in which the machinery operates.

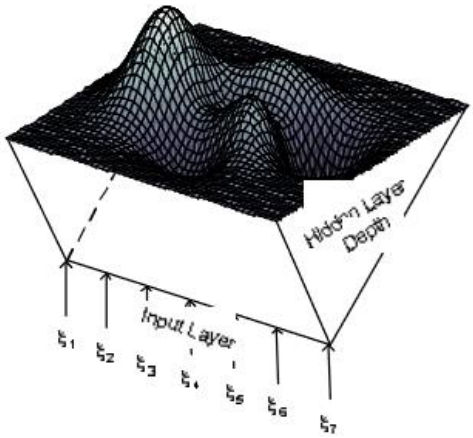


Figure 9 - Resulting cluster surfaces on the Self-Organizing Map (SOM). Source: Henze and Hindman (2002)

Nanayakkara et al. (2002) studied the applications of Artificial Neural Networks with dynamic activation function to control evaporator unit on an ammonia cooling system. Dynamic activation ANN are defined as feedback routines, wherein the outlet signal is feedback and combined to the inlet, resulting in a dynamic neuron and non-linear process – as it occurs on a static neuron. In such static cases, the inlet signal is manipulated by the neuron function weights and biases, goes through an activation function and is output. In their study, Nanayakkara et al. (2002) compared controlling an evaporator unit with static and dynamic activated neurons, determining that the former requires less time on the training stages, due to faster convergency, whilst achieving overall more effective control. Figure 10 presented the dynamic neuron utilized by Nanayakkara et al. (2002).

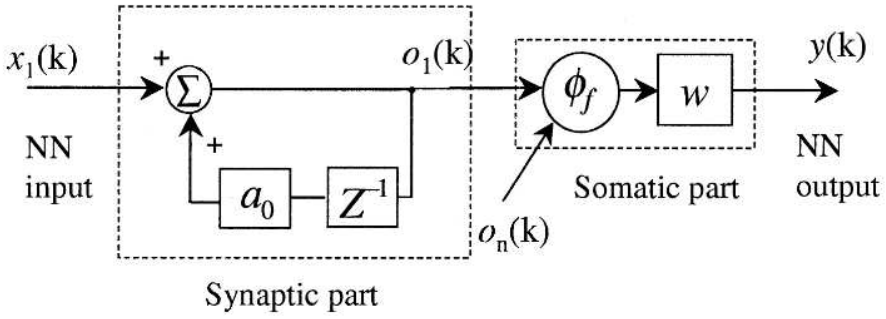


Figure 10 - Dynamic neuron utilized in Nanayakkara et al. (2002)

Zaheer-Uddin and Tudoroiu (2004) studied controlling an HVAC system discharge temperature with a PID controller tuned by a multilayer perceptron (MLP) Artificial Neural Network. In this research, the authors gathered readings from sensors installed throughout the

HVAC system (depicted in Figure 11) to feed the ANN, whose output determined the proportional, integral and derivative gains to the PID control and concluded that this type of adaptive tuning allowed a reduction in the stabilization time and less interference from external noises.

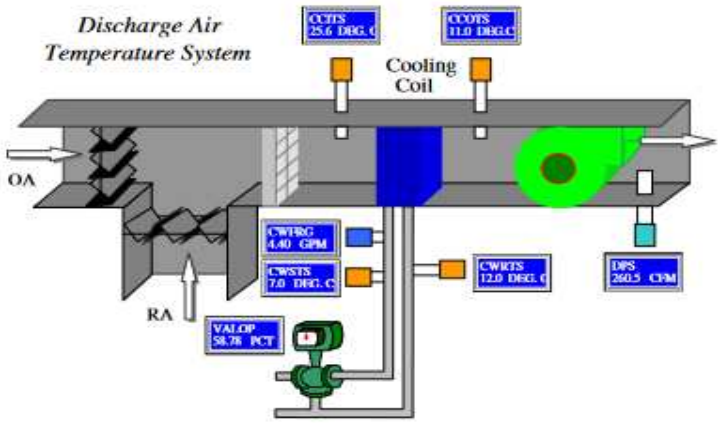


Figure 11 - HVAC plant controlled with ANN-PID contraption, Zaheer-Uddin and Tudoroiu (2004)

Abbassi and Bahar (2005) proposed controlling an evaporative cooler, presented in Figure 12, in an HVAC system parameters with a feedforward multiple layers ANN. By controlling the coolant mass flow rate, temperature, thermal load and relative humidity in the climatised regions, the authors identified and concluded that ANN are capable of achieving good results to PID control, by converging and minimising the steady state error faster than the conventional PID controller. The evaporator unit and ANN schematic representations are depicted in Figure *12*.

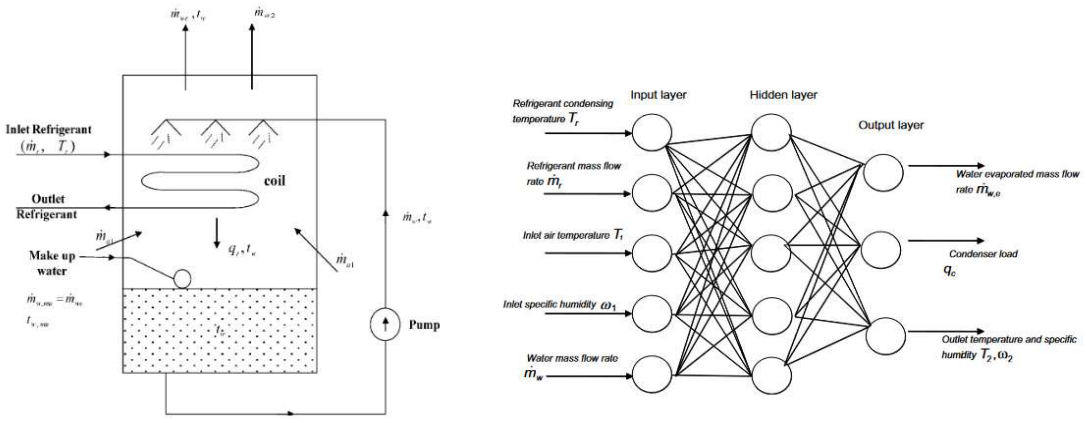


Figure 12 - Evaporator unit and ANN schematic depiction, from Abbassi and Bahar (2005)

Depending on the surrounding ambient conditions, automotive HVAC can introduce pollutant contaminated air into the climatized cabin. Thus, in their study, Lee and Chung (2005) implemented a single input, intermediate and output layer feedforward ANN, which took information from sensors which measured the climatized cabin air humidity and odor, whilst the output layer generated an air quality index. The neural network was implemented into an integrated microcontroller Atmega 128, which in turn, controlled the HVAC ventilation system, ensuring the cabin air quality

Wang and An (2006) e Wang et al. (2007) proposed controlling the temperature of climatised rooms in centralised unit HVAC systems. In their researches, the authors proposed a forward Multi-Layer Perceptron (MLP) ANN to tune PID controllers, by setting three neurons on the output layer, each responsible for the proportional (k_P), integral (k_I) and derivative (k_d) gain, respectively (depicted ahead in Figure 13). Both researches determined that the hybrid NEURO-PID (PID tuned by Neural networks) combination is better suited to effectively control the temperature than solely PID application, from the point of view of better control results. Moreover, Wang and An (2006) observed that NEURO-PID is less sensitive to external temperature changes, making it this, more robust and reliable and better suited for non-linear, applications with significant external perturbations. Wang and An (2006) also observed that the PID tuned with neural networks was able to reduce the overshoot by approximately 17% when compared to the traditional control.

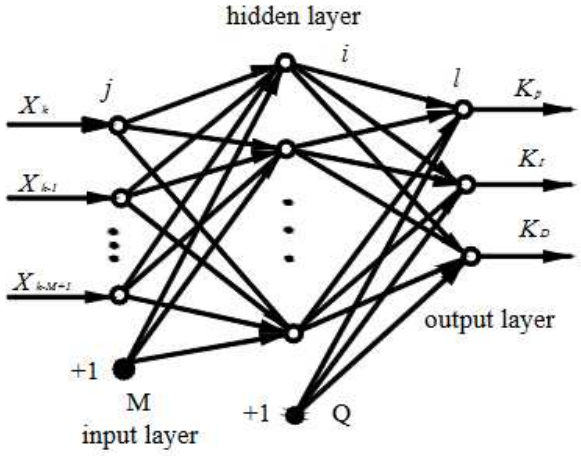


Figure 13 - Proposed Artificial Neural Network to tune a PID controlled (Wang and An, 2006)

Understanding the incident thermal load in an environment is important for establishing the best control logic in HVAC systems that will counteract it. In this context, Yao et al. (2006)

employed Radial Basis Function Neural Networks (RBFNN) to predict thermal load and consequently adjust the control system accordingly. The author was able to predict the thermal load satisfactorily with this technique when compared with a case study presented

Wang et al. (2007) employed a hybrid PID – Cerebellar Model Articulation Controller (CMAC). The CMAC is an ANN model first proposed by James Albus (1971) in which an associative memory holds the information required to adjust the networks weights and biases by establishing relations between inlet conditions and stores information. This stored information is originated from the ANN training process as well as previous operation experience. Wang et al. (2007) concluded that the integrated CMAC-PID is an effective control strategy, as it combines the PID excellent stability response to the CMAC fast responsivity. The CMAC-PID combination is depicted schematically in Figure *14*.

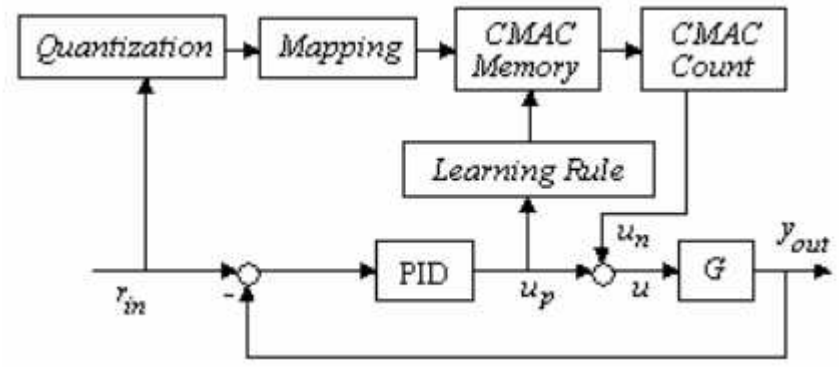


Figure 14 - Control strategies proposed by Wang et al. (2007)

Razi et al. (2006) proposed an automotive HVAC predictive control method to regulate the cabin air temperature. The proposed model operated by predicting the cabin air temperature based on tendencies observed from past measurements. This predicted value was then stored and compared to the actual future temperature measurement, and the error value used to adjust the ANN model. In this research, the author implemented a radial base activation function feedforward ANN with five neurons on the input layer, eighteen on the intermediate and one on the output. Finally, the authors compared the results from the ANN with conventional fuzzy-PID control and concluded that the ANN can perform the control interventions superiorly to the PID-fuzzy. A schematic view of the neuro-predictive automotive HVAC controlled diagram is presented in Figure 15.

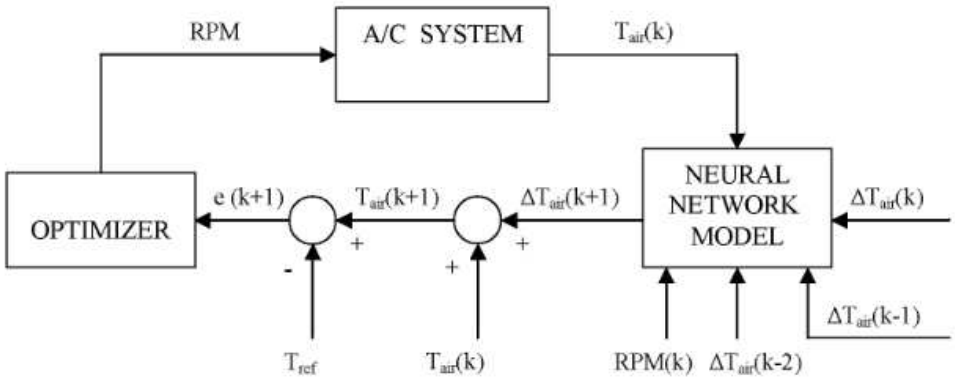


Figure 15 - schematic view of the neuro-predictive automotive HVAC controlled diagram (Razi et al., 2006)

Ensuring users thermal comfort depends on several parameters beyond simply controlling the dry-bulb temperature, such as humidity and airflow velocity. So, Torres and Martins (2008(a) and 2008(b)) proposed and trained an 8- input layer ANN that correlates thermal comfort related data measured in a HVAC system to an equivalent temperature, which was then input to a PID controller to achieve a better overall thermal comfort when compared to directly providing the measured temperature to the PID. The technique made it possible to satisfactorily predict the thermal comfort index when compared with theoretical values, causing the control to act to keep the PPD close to zero.

Guo and Song (2009) employed multilayers ANN integrated to a PI controller to control the volumetric flow rate into a generic climatised room. Khayyam et al. (2011) proposed an automotive cabin temperature control system based on the aforementioned ANN-PID interaction, in which the proposed ANN is composed of two neurons on the input layer, 10 on the intermediate and three on the output, each responsible for tuning the PID controller gain levels. The intermediate and outlet layers activation functions in this case were sigmoid and linear, respectively and the author achieved improvements in vehicle efficiency with this type of approach. This proposed architecture is depicted in Figure 16.

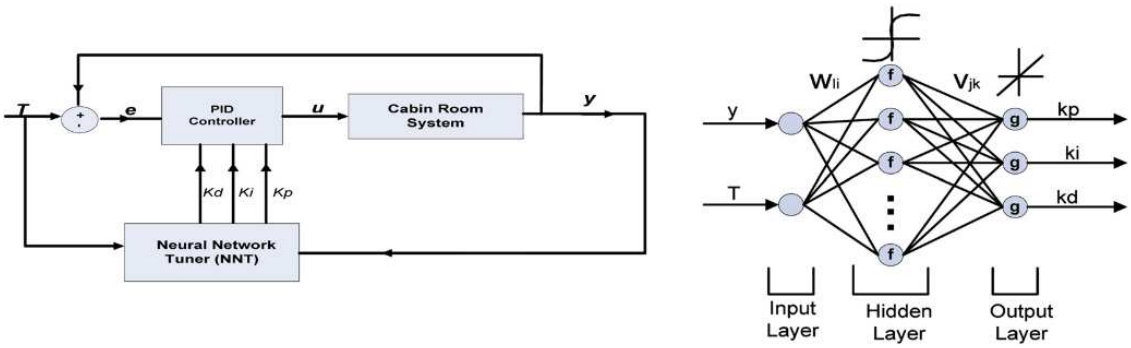


Figure 16 - Neuro PID (left) and Architecture (right) proposed by Khayyam et al (2011)

Kranz et al. (2012), the optimal evaporator unit blower rotational frequency and cabin air inlet flap opening degree were controlled by two sigmoid activation function feedforward ANN, each with eight neurons on the input layer receiving signal gathered from sensors throughout the vehicle. On the evaporator blower control ANN, the intermediate and output layers had 12 and 4 neurons respectively whereas the air inlet flap opening control is equipped with 10 and 3 neurons on the intermediate and output layers. The proposed ANN architectures are depicted in Figure 17.

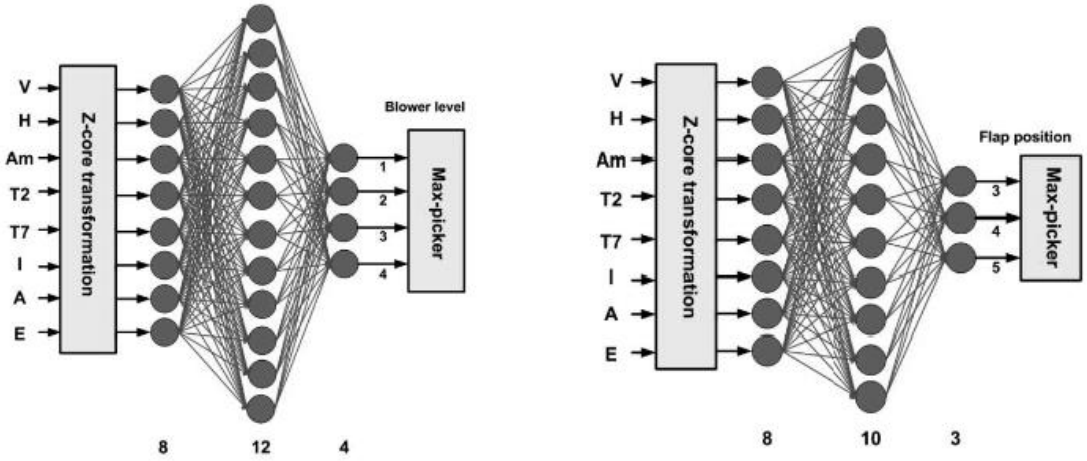


Figure 17 - Proposed ANN architectures, Kranz et al (2012)

Ferreira et al. (2012) proposed implementing radial base ANN to predict the predicted mean vote (PMV) index, which was then used as reference to control the climate of a room. Li et al. (2012) e Li et al. (2013) utilised in both publications a recurrent neural network to predict dry-and wet bult temperatures in a climatised space. Song et al. (2014) implemented a perceptron ANN to control and predict temperature changes in a computing data center.

Chiang et al. (2014) studied the application of Recurrent Neural Networks (RNN), or alternatively nonlinear autoregressive exogenous model (NARX), to predict the temperature in

an automotive cabin compartment as a function of the HVAC compressor rotational frequency. In this work, two alternative intermediate layer activation function were compared, sigmoid and radial base. This study concluded that it is possible to establish satisfactory control of the compressor rotational frequency based on the neural network temperature readings, and the sigmoid activation function is better suited to predict the temperature readings, albeit more sensitive to noise interference than the radially activated, particularly on the training stages. The proposed architecture is depicted in Figure 18.

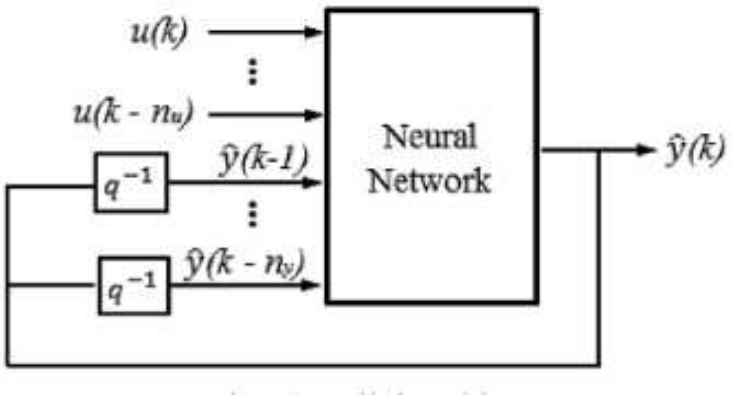


Figure 18 - Recurrent Neural Network proposed by Chiang et al (2014)

Ng et al. (2014) employed recurrent neural networks to represent the model of an MPC (Model Predictive Controller). In their work, trained the neural network using an experimental testbed. Subsequently, it was implemented an offline-trained model into the MPC controller, allowing for constant updates (online training). The authors compared this approach with MPC implementations that utilized only offline training, as well as applications that employed PID controllers. They concluded that the proposed application outperformed both other types of control.

Garnier et al. (2015) used Feedforward artificial neural networks to update the parameters of an MPC controller in an attempt to improve thermal comfort and reduce energy consumption, successfully achieving this strategy. Ku et al. (2015) used artificial neural networks in conjunction with Fuzzy logic to update the temperature setpoint of the controller, aiming to keep the thermal comfort index PMV around zero. Soudari et al. (2016) investigated the use of radial basis function neural networks to update the parameters of an MPC (Model Predictive Controller) based on occupants' behaviour, achieving energy savings ranging from 9% to 25% with this strategy.

In their work, Moon and Jung (2016) developed two Artificial Neural Networks, the first was trained to predict an HVAC optimal starting hour and the second to predict the energy consumption during the system operation, based on data measured on the climatised room to achieve thermal comfort. Attaran et al. (2016) implemented a Radial Base Function (RBF) architecture to tune a PID controller, aiming to improve an HVAC controlling capabilities. Yang et al. (2016) utilised a recurrent ANN to control the temperature and humidity levels in a climatised room. Javed et al. (2017) trained a feedforward ANN to control the heating, cooling and ventilation of a climatised space through cloud computing. X. Li et al (2017) proposed a recurrent Elman ANN to tune a PID controller, which was responsible for controlling the temperature of an ambient. Macarulla et al. (2017) employed feedforward artificial neural networks to predict the optimal timing of an MPC controller based on the internal ambient temperature, external temperature, and the temperature of a water-based heating system. With this strategy, they achieved a 19.69% reduction in energy consumption.

Png et al. (2019) studied applied ANN to predict thermal comfort integrated to numerous temperature measurements, aiming towards an appropriate ambient temperature control. Chaudhuri et al. (2019) utilised a feedforward ANN to control and maintain thermal comfort in residential buildings. Sun et al. (2020) proposed a feedforward ANN to predict a chiller water outlet temperature, and thus, control the air temperature in a climatised room through controlling the fan speed. Chen et al. (2020) employed a deep learning technique coupled to knowledge transfer to ensure thermal comfort is achieved in small buildings.

The new applications and technologies hint towards HVAC modern control techniques reliant on Deep machine learning, such as the Deep Reinforcement Learning (DLR). The DLR technique is based on the interaction between the ANN and the environment it is inserted, running with a multilayer dynamic structure to optimise the decision taking. The DLR topology is adapted and alters the decision taking agent, thus resulting in a new output action. This action interacts, then, with the programmed environment and the feedbacks are directed to the deep learning topology. Restarting the cycle. This cycling runs until the action matches the expected target observation, given the current environment. In Figure 19, an illustrative representation.

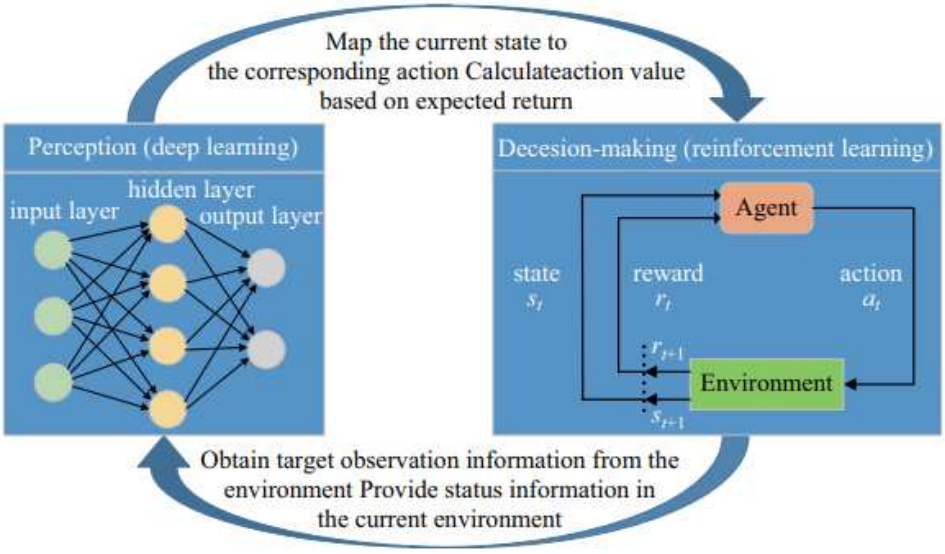


Figure 19 - Reinforcement learning (Zidong et al. 2020)

Zhiang Zhang and Lam (2018) employed an Asynchronous Advantage Actor-Critic (A3C) based approach for the control of radiant heating systems. The effectiveness of this method was validated in a real air conditioning system environment, with the proposed framework encompassing HVAC modeling, model calibration, training, and deployment. The results demonstrated a significant energy savings of 16.6% over a three-month period compared to Rule-Based Control (RBC).

Chen et al. (2018) introduced a Q-learning algorithm control strategy for optimal control decisions in HVAC systems, as well as for the opening and closing of windows. This strategy demonstrated a reduction in energy consumption ranging from 13% to 23% compared to rule-based control strategies. Brusey et al. (2018) proposed a control method based on reinforcement learning, wherein a passenger cabin thermal load model was developed and validated by experimental measurements provided by Hintea et al. (2014). With the validated model, the authors trained the algorithm to predict the cabin temperature, from which the evaporator unit blower rotation speed is controlled. Figure 20 in left, depicts the thermal load model proposed in the study whilst the right figure schematically depicts the interface between this thermal load model and the reinforcement learning algorithm.

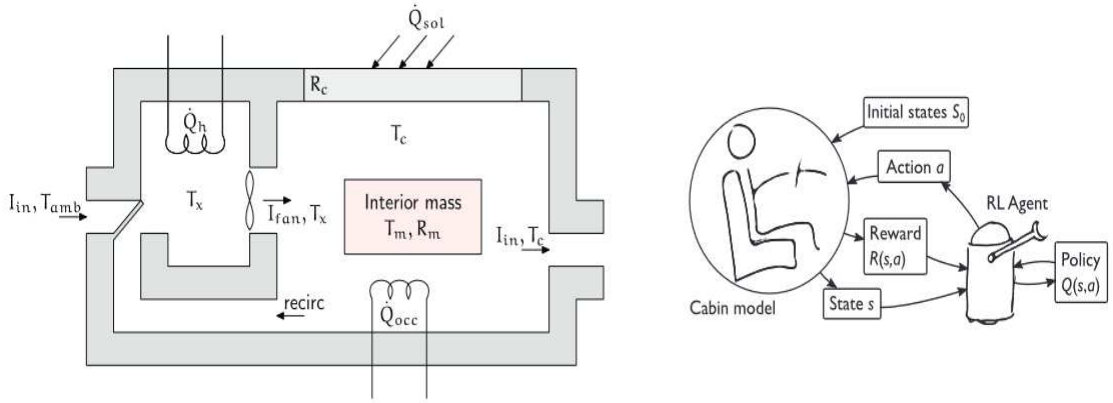


Figure 20 - Thermal load model proposed and interaction with the reinforcement learning ANN, Brusey et al. (2018)

Zhang et al. (2019) proposed a control framework based on deep reinforcement learning for a building energy model. The control framework consisted of four stages: building energy modelling, model calibration, deep reinforcement learning training, and control implementation. The control was implemented in a radiant heating system in an office building. A 78-day test revealed that, compared to the previous rule-based control, the new control strategy was able to save 16.7%.

Valladares et al. (2019) proposed a control method using Double Deep Q-Network algorithm to manage internal thermal comfort, air quality, and energy consumption in buildings. This approach demonstrated a more suitable Predicted Mean Vote (PMV) thermal comfort index compared to existing control systems, accompanied by a 10% reduction in CO_2 levels and a 4% to 5% decrease in energy consumption.

Wei et al. (2017) and Gao et al. (2020) implemented reinforcement learning to control the thermal comfort metrics in a residential zone. Gao et al. (2020) implemented a multilayer Perceptron to foretell the Predicted Mean Vote (PMV) as a function of temperature, humidity, Mean Radiant Temperature (MRT), airspeed velocity, occupants metabolism and solar irradiance. Upon training, the ANN was coupled to a reinforced learning algorithm responsible for controlling the ambient temperature and humidity levels as a function of the expected PMV and improvements arising form the intraction between the DRL and ambient conditions.

In Figure 21, a simple controller-HVAC application example is depicted, whereas on the mid subfigure, the trained neural network architecture used to predic the PMV, given the input data is depicted and, finally, the updated HVAC-ANN integration, acting upon the sensor gathered information. Both Gao et al. (2020) and Wei et al. (2017) concluded in their works that the hybrid Deep Reinforcement learning techiques is well suited to control residential

HVAC systems, capable of achieving a consideralby superior performance when compared to traditional control techniques.

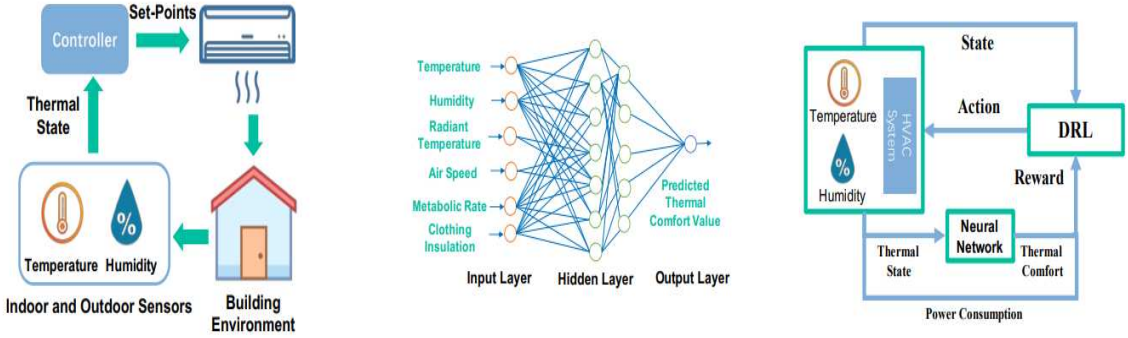


Figure 21 - Reinforcement learning Applied to residential HVAC control systems, Gao et al. (2020)

Brandi et al. (2020) conducted experiments to assess the adaptability of the Q-learning algorithm in the proposed control task across different scenarios. The results indicated energy savings between 5% and 12% when using the adaptive control agent compared to rule-based control logic.

Nagarathinam et al. (2020) propose a Multi-Agent Reinforcement Learning (MARL) algorithm aiming to reduce energy consumption in heating, ventilation, and air-conditioning (HVAC) systems while maintaining user comfort. This is achieved by adjusting the control of both the building and the chiller. To optimize training, they employ the transfer learning technique, where the learned network weights are used to initialize the various agents.

Qiu et al. (2020) proposed a Q-learning method for the control of the central chilled water system and optimization of the Coefficient of Performance (COP). The results of a three-month simulation indicated that the controller based on the Q-learning algorithm could save 7% to 11% in system energy consumption compared to basic and local feedback controllers. Zou et al. (2020) implemented a framework that utilizes Reinforcement Learning with Long Short-Term Memory (LSTM) recurrent neural networks. The DRL agents were able to achieve energy savings of 27% to 30% compared to actual energy consumption while maintaining the predicted percentage of discomfort (PPD) at 10%. These results underscore the effectiveness of the proposed framework in achieving optimal control, providing significant energy savings without compromising thermal comfort.

Biemann et al. (2021) conducted experiments evaluating the performance of four actor-critic algorithms for Variable Air Volume (VAV) control in a simulated data center

environment. These algorithms achieved a 10% reduction in energy consumption compared to model-based controllers while simultaneously maintaining zone temperature within the desired range.

Liu et al. (2021) utilised reinforcement learning techniques to improve an HVAC control system, pursuing a compromise between energy consumption decrease and enhance thermal comfort. In this paper, the authors concluded that by introducing the reinforcement learning ANN the energy related costs decreased whilst maintaining the appropriate thermal comfort levels. Y. Du et al. (2021) employed reinforcement learning techniques to directly control multiple thermally controlled residential zones and concluded that the ANN is particularly capable of adapting to numerous control conditions and has broad online applications potential.

In electric vehicles, the energy expended by the HVAC system has a greater impact than in combustion vehicles, given the lower amount of energy available for transportation in batteries compared to internal combustion engines. In an attempt to mitigate this issue, Choi et al. (2022) studied the application of the DQN (Deep Q Network) algorithm to control the rotation of the compressor and the fan of the evaporator. In their work, they used a reward function based on intermediate reward signals associated with the error between the cabin temperature and the setpoint, evaporator temperature, condenser pressure, and power consumption. They concluded that, compared to the rule-based method, there was a reduction in energy consumption.

Regarding electric vehicles, Haskara et al. (2022) employed Reinforcement Learning through the Q-Learning algorithm to adjust the setpoint of the HVAC system controller in an effort to maintain the thermal comfort index PMV within an acceptable range and thereby save energy. Joo et al. (2023) simulated temperature control in the automotive cabin using multiple agents in a Soft Actor-Critic Reinforcement Learning logic. One agent was responsible for controlling the compressor, while a second agent controlled the expansion valve. Through simulations and validations using the GT-SUITE R© software (Gamma Technologies, 2020), they concluded that the multi-Agent logic improved energy consumption by 53% in the transient control phase compared to a PID logic.

Fang et al. (2022) developed a multi-objective control strategy based on Deep Q-Network (DQN) for real-time adjustment of temperature setpoints in building HVAC systems. The primary goal was to balance energy consumption and internal thermal comfort. To evaluate this strategy, a co-simulation test environment using EnergyPlus-Python, following the FMI standard, was created. This allowed for easier assessment of the performance of advanced deep reinforcement learning (DRL) control algorithms in a simulated environment. Simulation

results indicated that the DQN strategy is effective in achieving a suitable balance between HVAC system energy consumption and internal air temperature.

Dai et al. (2023 -a) proposes a control strategy based on reinforcement learning to optimize air conditioning in commercial building systems. The proposed strategy employs the reinforcement learning method (Q-learning) to iteratively adjust controller parameters, aiming to achieve uniform cooling in different building zones. Validation tests demonstrate that the proposed strategy reduced the daily pre-cooling time by up to 12.1%, resulting in daily energy savings ranging from 5.1% to 17.8%.

Dai et al. (2023- b) explores the application of mixed-mode ventilation (MMV) in tropical climates characterized by high temperatures and humidity. Traditional natural ventilation is limited in these conditions, especially in areas with light winds. The research employs reinforcement learning to optimize the control of MMV, focusing on reducing energy consumption for cooling while ensuring occupants' thermal comfort. The study utilizes Deep Q-Network. Control variables considered include the window opening percentage and dynamic cooling setpoint temperature. Results demonstrate that RL can achieve a 52% reduction in cooling energy consumption compared to baseline methods

Qin et al. (2023) studied the algorithms of reinforce learning D3QN and PR-DQN for the control of Heating,cooling, and ventilation in an air conditioning (HVAC) systems in residential buildings. The optimization objective is to achieve the highest reward, considering thermal comfort and energy cost. The controller outperformed a rule-based control strategy, saving 63% in energy consumption with a 12.6% reduction in thermal comfort reward, resulting in an overall improvement of 15.3%. Solinas et al. (2024) studied the use of reinforcement learning in HVAC control tasks, finding positive results in terms of energy savings with the application of this technique.

Adesanya et al. (2024) used a flexible PID controller, optimized using a reinforcement learning algorithm (DRL) to effectively manage the temperature and energy consumption of the greenhouse. A TRNSYS-Python co-simulation framework was developed, allowing the DRL agent to interact in real-time with the greenhouse environment. The optimized PID parameters showed significant energy savings (8.81% to 12.99%) compared to on/off controllers and better temperature control (deviation of 2.07% to 3.13%) than manually tuned PID controllers.

Shin et al. (2024) used WDQN-temPER, a hybrid HVAC control method that combines a deep Q-network with a new technique of prioritized experience replay (PER) called temPER, and a Gated Recurrent Unit (GRU) model. The GRU model predicts future external temperature

and uses it as a state variable in the RL model. The temPER algorithm facilitates the use of samples in the RL training process with greater changes in external temperature based on the predicted temperature. Was experimentally demonstrated in EnergyPlus simulations that our WDQN-temPER model outperforms a rule-based model in terms of HVAC control, with energy savings of up to 58.79%. Table 2 provides a summary of the works, including the type of technology, control approach, control objective, and the type of training for artificial neural networks.

Table 2 - Summary of ANN Applied to conventional HVAC control system in literature

Reference	Type of technology	Control Approach	Control objective	Type of Training	Deep Learning use in HVAC
Teeter and Chow (1998)	Recurrent Neural Network	ANN-Offline	Temperature Control	Supervised	NO
M.Ueda (1999)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal comfort	Supervised	NO
Kajino et al. (2000)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal comfort	Supervised	NO
Henze and Hindman (2002)	Self-Organising Map	ANN-Online	Multi variables control	Unsupervised	NO
Nanayakkara et al. (2002)	Dynamic activation function neuron	ANN-Offline	Control an ammonia refrigerant evaporator	Supervised	NO
Zaheer-Uddin and Tudoroiu (2004)	Recurrent Neural Network and PID controller	Neuro-PID	Control of a discharge air temperature system	Supervised	NO
Abbassi and Bahar (2005)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Control of evaporative condenser	Supervised	NO
Lee and Chung (2005)	Microcontroller integrated Feedforward Multilayer Perceptron ANN	ANN-Offline	Air Quality Control	Supervised	NO
Wang and An (2006)	Multilayer Perceptron and PID controller	Neuro-PID	Controller in Central Air-conditioning System	Supervised	NO
Razi et al. (2006)	Radial activation function Recurrent Neural Network	ANN-Offline	Temperature Control	Supervised	NO
Jiangjiang Wang et al. (2006)	Multilayer Perceptron and PID controller	Neuro-PID	Controller in a Single-zone HVAC System	Supervised	NO
Wang et al. (2007)	CMAC-PID	Neuro-PID	Controller in HAVC	Supervised	NO

Continue...

Torres and Martin (2008)	Neuron – PID hybrid	ANN-Online	Thermal Comfort	Supervised	NO
Guo and Song (2009)	Multilayer Perceptron and PID controller	ANN-Online	Air pressure control performance of variable air volume (VAV) system.	Supervised	NO
Ferreira et al. (2012)	Radial basis RNN	ANN-Offline	Thermal Comfort	Supervised	NO
Li et al. (2012)	Feedforward Multilayer Perceptron ANN	ANN-Online	Temperature control of a air conditioning system	Supervised	NO
Kranz et al. (2012)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal comfort	Supervised	NO
Li et al (2013)	Feedforward Multilayer Perceptron ANN	ANN-Online	Temperature control of a air conditioning system	Supervised	NO
Chiang et al. (2014)	Recurrent neural network	ANN-Offline	Temperature Control	Supervised	NO
Song et al. (2014)	Feedforward Multilayer Perceptron ANN	Neuro-PID	Temperature control of Data Center	Supervised	NO
Ng et al. (2014)	Recurrent neural network In MPC Control	ANN-Online	Temperature Control	Supervised	NO
Ku et al.(2015)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal Comfort	Supervised	NO
Garnier et al. (2015)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal Comfort and Temperature control	Supervised	NO
Soudari et al. (2016)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal Comfort and Temperature control	Supervised	NO
Moon and Jung (2016)	Feedforward Multilayer Perceptron ANN	Logic coupled with ANN	Thermal Comfort	Supervised	SIM
Attaran et al. (2016)	Neuron – PID hybrid and RBF architecture	Neuro-PID	Control of humidifier and heating coil	Supervised	NO
Yang et al. (2016)	Recurrent Neural Network	ANN-Offline	Control of indoor air temperature and humidity	Supervised	NO

Continue...

Javed et al. (2017)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal Comfort	Supervised	NO
X. Li et al (2017)	Neuron – PID hybrid and Elman Recurrent Neural Network	ANN-Offline	Control for indoor temperature time-delay	Supervised	NO
Wei et al (2017)	Reinforcement Learning – Deep Q-Network	Reinforcement Learning	Temperature control	Reinforcement Learning	NO
Macarulla et al. (2017)	Multilayer Perceptron	ANN-Offline	Temperature control	Supervised	NO
Zhang and Lam (2018)	Reinforcement Learning- Asynchronous Advantage Actor Critic	Reinforcement Learning	Control of indoor air temperature, Thermal Comfort and save energy	Reinforcement Learning	NO
Brusey et al. (2018)	Reinforcement learning- SARSA	Reinforcement learning	Thermal comfort	Reinforcement learning	NO
Chen et al. (2018)	Reinforcement Learning – Q-Learning	Reinforcement Learning	Control of indoor air temperature, Thermal Comfort and save energy	Reinforcement Learning	NO
Brandi et al. (2020)	Reinforcement Learning – Deep Q-Network	Reinforcement Learning	Control of indoor air temperature.	Reinforcement Learning	NO
Valladares et al. (2019)	Reinforcement Learning – Double Deep Q- Network	Reinforcement Learning	Control of indoor air temperature, Thermal Comfort and save energy	Reinforcement Learning	NO
Nagarathinam et al. (2020)	Reinforcement Learning- Multi-Agent Reinforcement Learning (MARL)	Reinforcement Learning	Control of indoor air temperature, Thermal Comfort and save energy	Reinforcement Learning	NO
Sun et al. (2020)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Control method of chilled water temperature	Supervised	NO
Png et al (2019)	Cloud integrated Feedforward Multilayer Perceptron ANN	ANN-Online	Thermal Comfort	Supervised	NO
Chaudhuri et al. (2019)	Feedforward Multilayer Perceptron ANN	ANN-Offline	Thermal Comfort	Supervised	NO
Zhang et al. (2019)	Reinforcement Learning- Asynchronous Advantage Actor Critic	Reinforcement Learning	Control of indoor air temperature and save energy	Reinforcement Learning	NO
Chen et al. (2020)	Learning transfer ANN	ANN-Offline	Temperature control	Supervised	SIM

Continue...

Gao et al. (2020)	Reinforcement Learning	Reinforcement Learning	Thermal Comfort	Reinforcement Learning	NO
Qiu et al. (2020)	Reinforcement Learning – Q-Learning	Reinforcement Learning	Thermal Comfort and save energy	Reinforcement Learning	NO
Zou et al. (2020)	Reinforcement Learning – Deep Deterministic Policy Gradient (DDPG	Reinforcement Learning	Thermal Comfort and save energy	Reinforcement Learning	NO
Liu et al. (2021)	DDPG - Reinforcement Learning	Reinforcement Learning	Energy Control	Reinforcement Learning	NO
Y. Du et al. (2021)	DDPG - Reinforcement Learning	Reinforcement Learning	Control of indoor air temperature and save energy	Reinforcement Learning	NO
Biemann et al.(2021)	SAC, TD3, PPO, TRPO – REINFORCEMENT LEARNING	Reinforcement Learning	Control of indoor air temperature and save energy	Reinforcement Learning	NO
Fang et al. (2022)	Reinforcement Learning – Deep Q-Network	Reinforcement Learning	Control of indoor air temperature and save energy	Reinforcement Learning	NO
Choi et al (2022)	Reinforcement learning – Deep Q Network	Reinforcement learning	Temperature and Thermal comfort control	Reinforcement learning	NO
(Haskara, Hegde, and Chang 2022)	Reinforcement learning – Q-Learning	Reinforcement learning	Temperature, Thermal comfort control and save energy	Reinforcement learning	NO
Dai et al. (2023 -a)	Reinforcement Learning – Q Learning	Reinforcement Learning	control of indoor air temperature and save energy	Reinforcement Learning	NO
Dai et al. (2023 -b)	Reinforcement Learning – Deep Q Network	Reinforcement Learning	Thermal Comfort and save energy	Reinforcement Learning	NO
Du et al. (2021)	Reinforcement Learning- Deep Deterministic Policy Gradient (DDPG)	Reinforcement Learning	Temperature control	Reinforcement Learning	NO
Qin et al. (2023)	Reinforcement Learning - D3QN and PR-DQN	Reinforcement Learning	Temperature and humidity control	Reinforcement Learning	NO
Joo et al (2023)	Reinforcement learning - Multi-Agent Reinforcement Learning (MARL)	Reinforcement learning	Temperature control and save energy	Reinforcement learning	NO
Shin et al. (2024)	Reinforcement Learning	Reinforcement Learning	Temperature control and save energy	Reinforcement learning	NO
(Adesanya et al. 2024)	Reinforcement Learning	Reinforcement Learning	Temperature control and save energy	Reinforcement learning	NO
Solinas et al. (2024)	DDPG - Reinforcement Learning	Reinforcement Learning	Energy Control	Reinforcement Learning	NO

Continue...

2.4 Closure for literature review

This chapter presented the fundamentals of artificial intelligence in sections 2.1 to 2.4, with a primary emphasis on recurrent neural networks and backpropagation through time, which is how recurrent networks learn. Additionally, an extensive literature review was conducted on the applications of artificial neural networks in the control of HVAC systems, whose conclusions on the state of the art will be presented in this chapter's closing remarks.

From the works listed in Table 2, only Moon and Jung (2016) and Chen et al. (2020) used deep learning algorithms; however, not for tuning the gains of the PI/PID controller. Additionally, none of them used deep learning with recurrence among neurons, highlighting the gap to be filled.

Artificial neural networks applied in HVAC control is a current challenge for the field of refrigeration and air conditioning. From a control perspective, overall, thermal management of an environment is a challenging task, as comprehensive modeling may require the consideration of over 1000 equations, as noted by Wang et al. (2007) in their study. Furthermore, disturbances, such as the opening and closing of doors and windows, among other factors, can introduce randomness into the air conditioning operation

The applications of artificial neural networks in HVAC aim to overcome the challenge of modeling the thermal behavior of an environment. By collecting data and subsequently training the network, it can understand the dynamics of the system even if it is not explicitly modeled. However, several authors note that the exclusive and isolated use of neural networks is limited to situations for which they were trained, restricting their capacity for abstraction and generalization

To mitigate the issue of artificial neural networks being limited to the training data domain, many authors have chosen to combine Machine Learning with PID or PI algorithms, creating a control algorithm commonly referred to as Neuro-PID or Neuro-PI. This approach allows for establishing a level of control even under unforeseen circumstances, as any errors introduced by the neural network are absorbed by the PID/PI control formulation

However, the use of Neuro-PID is not universally applicable. In scenarios where the system exhibits considerable delay, the use of Neuro-PID becomes unfeasible. Ideal conditions for employing Neuro-PID arise in systems where the controllability ratio, defined as the ratio of delay to time constant, is less than 2 (Salavati, Grigoriadis, and Franchek 2022). Additionally, when the control objective is thermal comfort, it becomes essential to regulate both the ambient temperature and relative humidity, among other parameters. Temperature and Humidity variables cannot be effectively controlled independently by two PID algorithms due

to their strong coupling, and attempts in this direction often lead to unsatisfactory results, as mentioned by N. Li et al. (2012).

For situations where Neuro-PID control is not suitable, it is possible to directly use neural network control. In cases where disturbances in the system are problematic enough that the Machine Learning algorithm does not perform correctly, it is still possible to utilize the technique with online operation. In online operations, the neural network is initially trained offline to start operation. However, data collection remains constant, and over time or due to a system event, the network can be retrained automatically, allowing the Machine Learning algorithm to adapt to this new, unforeseen situation. This possibility has emerged recently with the advent of cloud operations and the Internet of Things (IoT), as observed in the work of Javed et al. (2017).

In addition to the option of online training and cloud-based operation, for situations where Neuro-PID is not viable, it is possible to use Machine Learning algorithms coupled with specific control logics or even employ various Machine Learning algorithms for different scenarios. An example presented is that of Moon and Jung (2016), who utilized two neural networks, one for situations when the environment is sparsely occupied and another for when the environment is densely occupied.

In light of the challenges posed by the inherent non-linearity of thermal systems, some authors have started to adopt reinforcement learning techniques, as demonstrated by Brusey et al. (2018). In this type of technique, you don't have input and output data to train the model. The algorithm operates based on programming the thermal environment to be controlled. In this programming, you can make predictions about random events, such as window openings or unexpected thermal loads in traffic scenarios. The algorithm involves an agent that learns the best way to operate to achieve a specific objective even in the presence of random events.

Considering the information presented in the previous paragraphs, a logical approach for selecting the use of artificial neural networks in HVAC system control is suggested in Figure 22. First, it is determined whether the HVAC goal is thermal comfort or the control of a single variable, such as temperature. If it's thermal comfort, both relative humidity and temperature should be controlled, and due to the strong coupling between humidity and temperature, PID control is not advisable. Next, the issue of noise is considered. If the system is very noisy, such as in automotive cabins, the option of using artificial neural networks in isolation may not be the best choice, given their limited applicability to situations they were trained for. Therefore, for such situations, it is suggested to use online-trained neural networks, reinforcement learning, or even consider artificial neural networks acting in conjunction with other control logic

structures. If the system is not noisy, the use of offline-trained artificial neural networks may be sufficient.

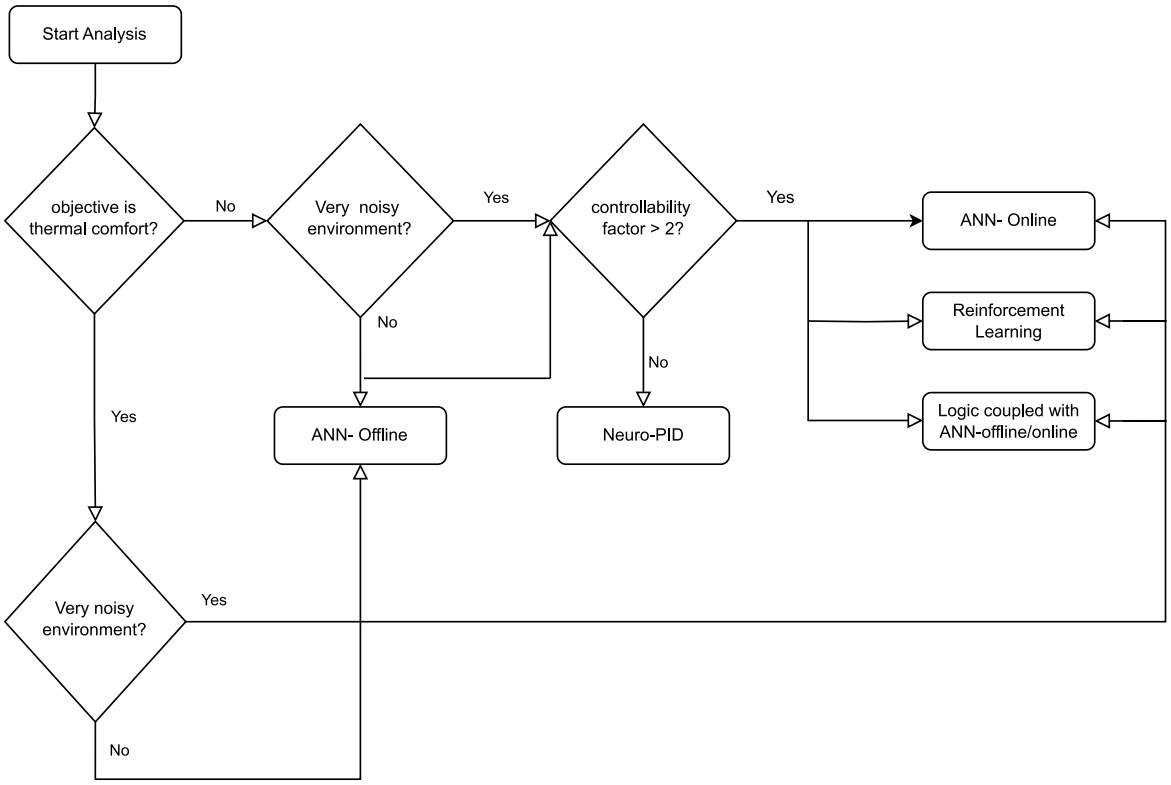


Figure 22 - Machine Learning Approach in HVAC Control. Source: (Mendes et al. 2024)

If the goal is not thermal comfort, it is possible to use neural networks with offline training for variable control as long as the system is not very noisy. Otherwise, if the system has a controllability factor less than 2, the use of artificial neural networks for PID controller tuning can be considered. This option can also be considered if the system is not very noisy. For cases where the controllability factor is greater than 2, options like online-trained neural networks, reinforcement learning, or even ANN coupled with other control logics are suggested. The thermally controlled environment discussed in this thesis does not aim for thermal comfort and is not noisy. It also does not have a controllability factor greater than 2, so the use of the Neuro-PI algorithm for it is feasible.

Considering the review of the state of the art presented and the articles included in Table 2, the types of training employed in artificial neural networks for HVAC control are presented in Figure 23. Thirty-seven percent of the approaches used databases that allowed training neural networks at once (offline) to directly operate HVAC systems, while 9% used databases for online training. Twelve percent chose to use artificial intelligence coupled with PI or PID

controllers. Two percent employed artificial neural networks in conjunction with some form of control logic, such as using one network to control the system in a populated environment and another network in an empty environment. Finally, 40% of the studies used Reinforcement Learning as the control technique for HVAC systems.

Figure 23 is a pie chart divided into two halves. The right half shows the percentages of works based on the types of neural network training: supervised, unsupervised, or reinforcement. The left half presents the control approaches used by researchers, in accordance with Figure 23. It can be seen that Neuro PI control is among the least common, with only 12% of the works listed in Table 2.

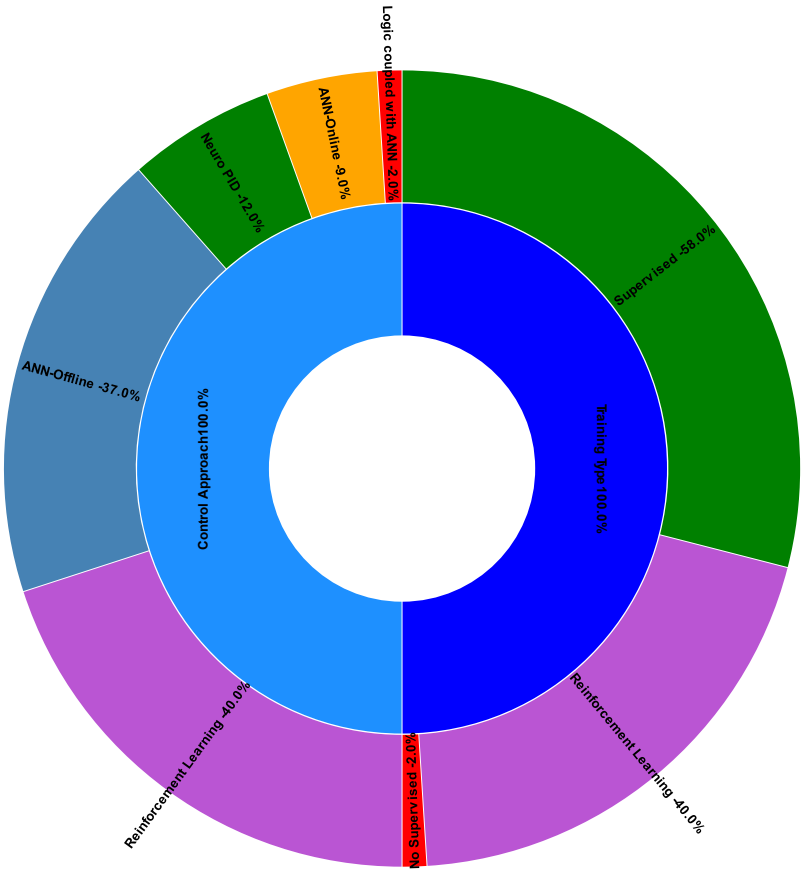


Figure 23 - State of the art: percentages. Source: (Mendes et al. 2024)

Considering the gap in the use of deep learning for adaptive control of the PI controller, the next chapter will present the methodology that allowed us to conduct two analyses: one using deep learning to tune the PI controller in a simulated Python environment, considering various plants from the work of Yamazaki et al. (2011); and another regarding the experimental application of deep learning for HVAC control.

3 METHODOLOGY

Most control systems do not operate on an environment consisting of a single plant but rather a variety of models, each presenting distinct characteristics and performances (Aguirre, 2007). In the context of environments conditioned by HVAC systems, the controlled thermal zones vary. That is, temperature response variations occur due to changes in incident thermal loads, variations in supplied airflows, air infiltration, and changes in the opening or closing of doors and windows, among other factors.

Given the presence of multiple plant models in the system, the methodology of this thesis begins with determining these models based on parameters that affect thermally controlled environments. Next, the closed-loop transfer functions with unit feedback are considered for stability analysis of each model. If the system is considered stable, the controller gains associated with the plant are recorded in a database for later training of artificial neural networks for adaptive identification in Neuro-PI control.

In the simulation, the work of Yamazaki et al. (2011) was used as the basis for constructing the twenty-four plants considered in Table 4 to be presented during this section. All plants had variations in static gain, time constant, and transport delay parameters compared to the parameters experimentally established by Yamazaki et al. (2011). Changes in plant parameters over the simulation time were thought of as changes that could be caused by variations in the mass flow rate of the blown air. Thus, the controller gains were calculated using correlation of Viziole's (2003) for each plant and were tuned by neural network models based on two mass flow rate inputs, which are the return air and recirculation air.

Regarding experimental implementation, the parameters of static gain, time constant, and transport delay were determined experimentally for two plants. One plant was associated with a thermal load of 40 W acting on a climatized environment prototype and the other with a thermal load of 100 W, these thermal load values arise from operational restrictions of the test bench. Based on these parameters, the controller gains for each plant were calculated.

In the training phase of the artificial neural networks, was utilized deep recurrent networks, both unidirectional and bidirectional, LSTM networks with and without bidirectionality, as well as deep feedforward networks. The training process was conducted in Python using the TensorFlow library (Paul Barham et al., 2015) and the Keras API (Chollet, F. 2015). During this process, various models with different architectures and hyperparameters were monitored using the Mlflow library (Databricks, 2018) . Appendices 9.1, 9.2, 9.3, and 9.4

contain the model training codes, and Annex 9.5 contains the code for creating the machine learning experiment within Mlflow to evaluate the best model architecture.

The setup of an HVAC system was built, where an environment with two possible thermal loads had its temperature controlled at setpoints of 21, 22, 23, and 24ºC. The adaptive PI control was implemented on a Raspberry Pi 4 Model B, which received data via HTTP communication with computers in the AWS Cloud and via FTP from the data acquisition system. This experimental setup allowed for comparing the simulation results and discussing real-world applications of deep learning algorithms in adaptive control. Figure 24 provides an overview of the methodology of this work.

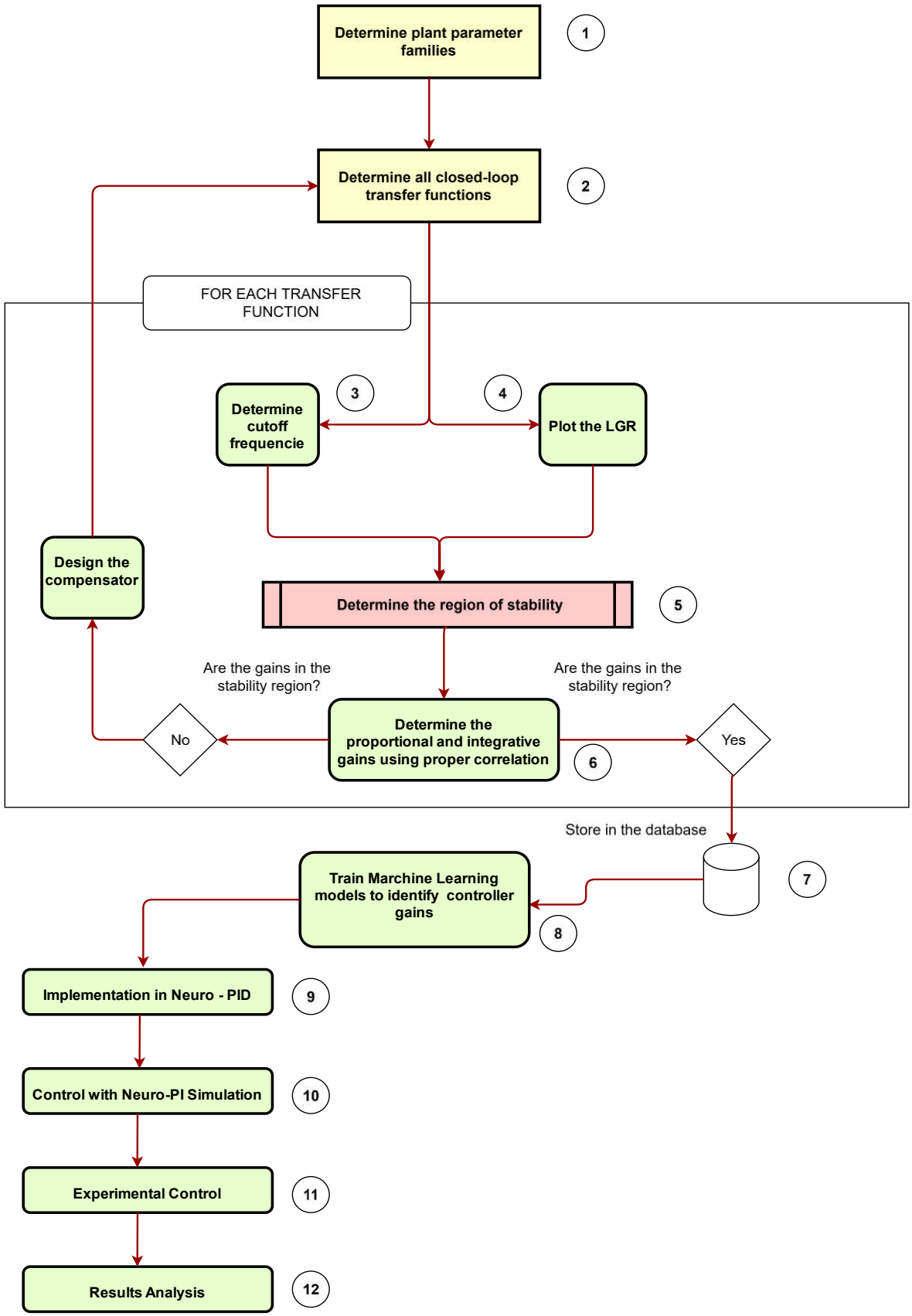


Figure 24 - Flowchart methodology for Steps developed in this thesis

Section 3.1 will discuss how the parameters of the simulated and experimental plants were obtained. Section 3.2 will present the method for evaluating the stability of the control concerning the previously obtained parameters. Section 3.3 will show how the neural networks were trained to identify the controller gains for the simulated and experimental plants. Section 3.4 will present the Neuro-PI control. Sections 3.5 will present the Python environment where the Neuro-PI control was simulated, as well as a more detailed explanation of the simulation. Items 3.6 and 3.7 will discuss the experimental setup and the cloud control systems project that allowed the experimental operationalization of the work.

3.1 Plant identification

The criterion used to build the simulated environment plants was initially based on the study by Yamazaki et al. (2011), which addressed the air conditioning control of a residence with a PID controller. The plant proposed by Yamazaki et al. (2011) served as a starting point, but with one modification: the static gain (K) was adjusted to be negative, representing a cooling process instead of heating.

In Yamazaki's study, the plant parameters were used as a baseline to generate 8 plants by varying the three components (K, τ, θ). For each of these 8 plants, simulations were conducted where the mass flow rate of renewal and recirculation air was increased at two specific times (3000 and 6000 seconds into the simulation). This means that for each of the eight plants, there are now two additional variations due to the changes in simulated air mass flow rate. Figure 25 illustrates the plants variation considered in the simulation.

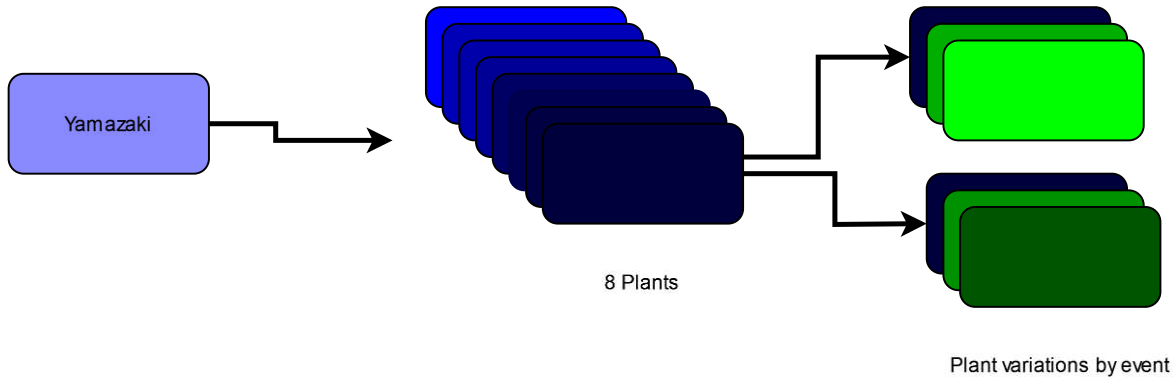


Figure 25 - Plants in Simulation case

Thus, deep learning algorithms were used to update the controller gains according to the ventilation within the environment. A more detailed explanation of the simulation, as well as

the values of the plant parameters considered and the static gains for each of them plants, is provided in section 3.5.

Regarding the experimental part, the plants were identified based on the incident thermal load. Given the two types of thermal loads, the control will therefore act on two distinct plants: one associated with a thermal load of 40 W and the other associated with 100 W. The manipulated variable was the frequency of the three-phase motor, and to identify the plants, a step change in the rotation frequency of 27 Hz was performed through the PWM control of the Raspberry Pi. From the results, it was possible to establish the plant's static gain (K), the time constant (τ), and the transport delay (θ) for both situations. The controller gains were calculated according to the methodology presented in section 3.2, and with these, the neural networks were trained according to the methodology presented in section 3.3. Figure 26 presents the step response results in the plant identification process for 40 W and 100 W of thermal load. As well as presenting a schematic figure of the experimental setup.

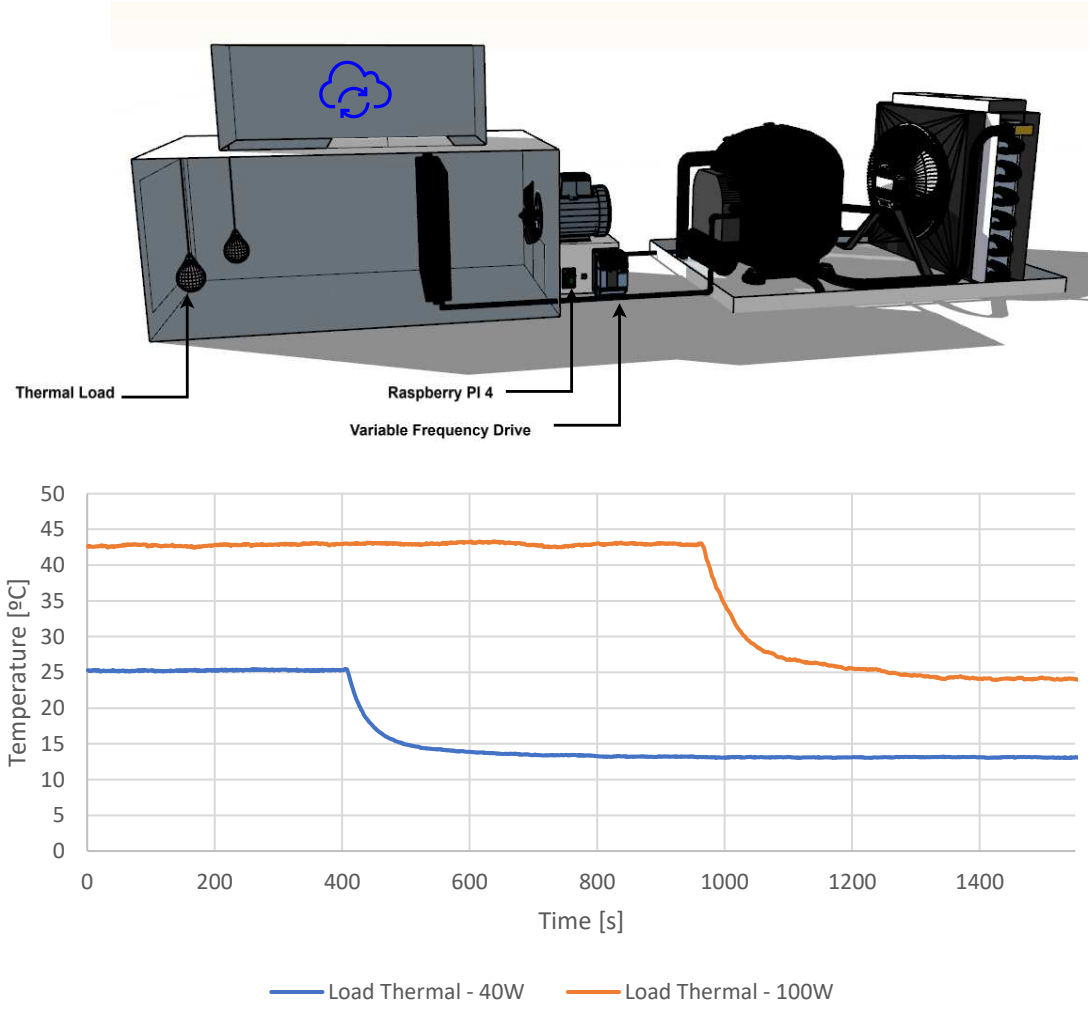


Figure 26 – Experimentl Setup and Step response graph for plant identification

For the plant associated with a 40 W thermal load, the static gain (K), time constant (τ), and transport delay (θ) were -0.450099, 17s, and 7s respectively. For the plant associated with 100 W, they were -0.88803, 24s, and 7s. These values are very close to those found by Yamazaki et al. (2011) in your experimental work and used during the simulation stage, where static gain was -0.64, time constant was 18s and delay time was 2.4s. The proportional and integral gains for the 40 W plant were -0.55742 and -0.02291, and for the 100 W plant were -0.25876 and -0.008268, all calculated following Viziole (2003).

3.2 Stability analysis for multiple plants

Temperature variation in an environment can be represented as a first-order model with transport delay (FODPT) (Stoecker and Stoecker, 1989). However, the parameters of this model, process gain (K) - time constant (τ) and transport delay (θ), are variable. And, in PI control applications, the characteristics of the plant to be controlled directly influence the closed-loop stability.

Regarding the PI controller, the control signal consists of a proportional gain (kp) and an integral gain (ki) that can be tuned according to the plant parameters by different methods, such as Ziegler and Nichols (1993), Cohen and Coon (1953) , Internal Model Control (IMC) (Rivera, Morarl, and Skogestad 1986), Optimum Integral Error methods for disturbance loads (ISE-load, ISTE-load) (Ho et al. 1996), Modified Anti-Windup (Visioli 2003), among others.

From a stability point of view, Tan's work (2004) presents a methodology to verify the controller gains values that make the control stable. The theory proposed by the author is based on the position of the gains in stability regions for closed-loop control. These regions, in which the proportional and integral gains values will be positioned, are determined by Equation. 25 and Equation 26.

$$kp = \frac{(\omega^2 \cdot N_0 \cdot D_0 + N_e \cdot D_e) \cdot \cos(\omega \cdot \theta) + \omega \cdot (N_0 \cdot D_e - N_e \cdot D_0) \cdot \sin(\omega \cdot \theta)}{-(N_e^2 + \omega^2 \cdot N_0^2)} \quad (25)$$

$$ki = \frac{\omega^2 \cdot (N_0 \cdot D_e - N_e \cdot D_0) \cdot \cos(\omega \cdot \theta) - \omega \cdot (N_e \cdot D_e + \omega^2 \cdot N_0 \cdot D_0) \cdot \sin(\omega \cdot \theta)}{-(N_e^2 + \omega^2 \cdot N_0^2)} \quad (26)$$

Where ω is the frequency, N_0 is the imaginary part of the plant's numerator, N_e is the real part of the plant's numerator, D_0 is the imaginary part of the plant's denominator, D_e is the real part of the plant's denominator, and θ is the time delay. The frequency can vary from zero

to infinity, however, it is advisable that the stability region be defined for values below the cut-off frequency. In Tan's work (2004) there are explanatory examples of the method. After the stability region is determined, the controller gains are determined by a suitable correlation and it is verified whether these gains are within this region. In this work was use the Modified Anti-Windup (Visiole, 2003). The Visiole (2003) correlation was used due to its ability to mitigate the windup error of the PI controller and is implemented in the Matlab PID Tuning Toolbox (2022) presented in Figure 27. By using a correlation capable of mitigating errors associated with transient characteristics of the controller, any improvement in transient control using neural networks becomes even more relevant.

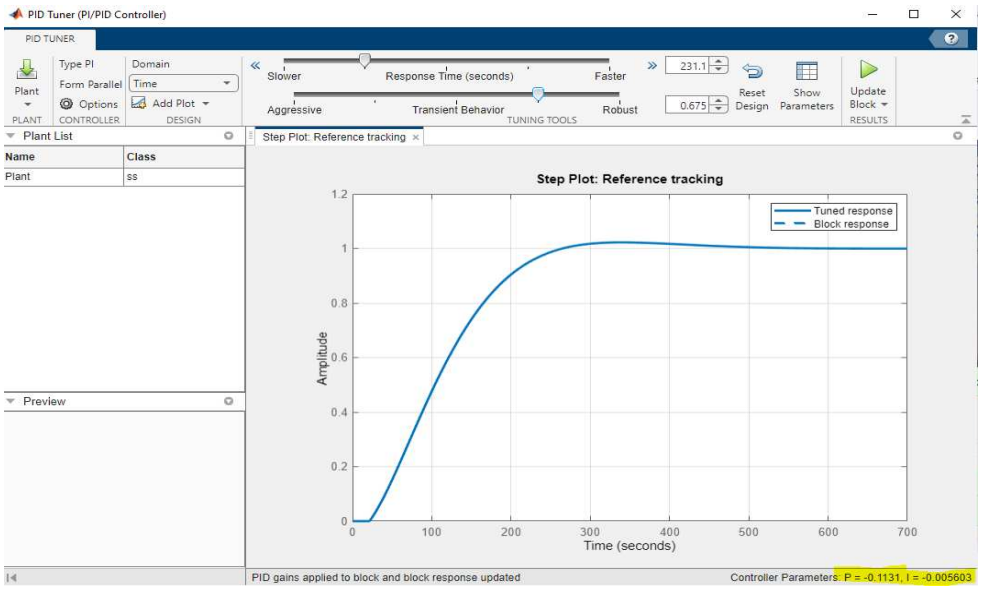


Figure 27 - PID Tuner Matlab

This methodology is repeated for each transfer function identified for each of the multiple plants for both the simulation and experimental plants. Generating a database with values of controller proportional gain and integral gain in function of the plants behavior. For neural network training, in experimental analysis the incident thermal load is considered as input and in simulation analysis the flow mass is considered as input, in the booths analysis, the proportional and integral gains is considered as outputs; in the sequence the gains are stored in the database after the stability check.

3.3 Training of neural networks

The neural networks were trained using Python programming, using the TensorFlow frameworks and the Keras API. Several models were trained, varying the number of

intermediate layers from 2 to 8. The number of neurons in each layer varied from 16 to 256, also considering dropout with rates of 10%, 20% and 50%, in addition to the Hyperbolic Tangent and Rectified Linear Unit (Relu) activation functions. For each time step and for each type of neural network, the best architecture was determined. The monitoring of the best architecture was carried out using the MLFLOW library, which selects the models that present the lowest mean squared error (MSE) and whose code is presented in appendix 5.

Both for deep recurrent neural networks and LSTM recurrent neural networks, with and without bidirectionality, the analysis allowed us to conclude that the most suitable architecture is composed of two intermediate layers, with a 20% dropout between the last intermediate layer and the output layer. The layers have 128 neurons each for the simple recurrent neural networks. For LSTM networks, 128 LSTM cells were stacked with the standard internal structure of each.

Regarding the Deep Feedforward network, the best architecture evaluated has 360 neurons in each of the two intermediate layers. The Relu function was used in these. A 20% dropout was used between the penultimate intermediate layer and the output layer. The linear activation function was adopted for the output layer of both the Feedforward and recurrent networks, with 2 neurons being used in the output layer, representing the proportional gain and the integral gain.

Regarding the simulation, the database was built considering a parametric variation in the process at the approximate times of 3000 and 6000 seconds. This represents changes in the plants due to changes in the return and renewal air, also resulting in changes in the controller gains, which were stored in the database for training the neural networks. The mentioned variation and the construction of the database will be better presented by Table 4 in the simulation topic.

For the training of recurrent neural networks, a time step of 10 seconds was considered. In other words, for example, the recurrent neural networks in simulation will start the tuning process at seconds 2990, unlike the Deep feedforward neural networks, which perform adaptive tuning at the exact moment of the event, i.e., at approximately 3000 seconds. The time step is characteristic of recurrent networks and needs to be informed beforehand in the training phase, as their predictive capacity is within this time interval.

As described in item 2.2.1, recurrent networks unfold their neurons according to the time step; was considered a time step of 10 seconds. Figure 28 shows the tensor formed by the input data with 10-time steps and the output for each sample used. Recurrent networks take into account the entire trend presented within each internal matrix of 10 data points to make the

output prediction, which differs from deep feedforward networks that would consider not a matrix but a vector with a single input associated with a single output.

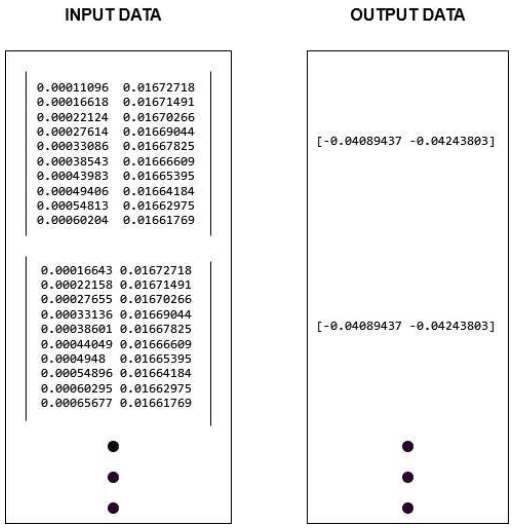


Figure 28 - Tensor Data Recurrent Neural Network

In all trained neural networks, the mean squared error (MSE) is below 0.000002. The initial learning rate was 0.001 and was reduced by 0.97 every 100 epochs with no change in MSE. The batch size was 800 data, and they were regularized using L2 regularization. The number of epochs varied for each type of network until the MSE was less than 0.000002, and the gradient descent method used was Adam (Ruder 2016).

The Figure 29 shows the monitoring of the models by MLFLOW. On the left side are the names of the various models analyzed. On the upper right side, is seen a range of MSE values. On the lower left side, there is a graph showing the gradient descent for each model, and on the lower right side, the models that presented the lowest MSE values are displayed.

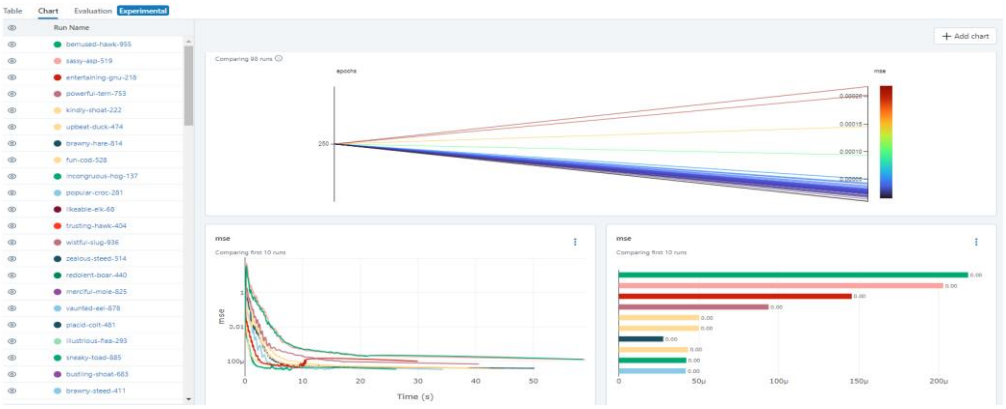


Figure 29 - Dashboard for monitoring models with MLFLOW experiment

Regarding the experimental dataset, during the neural network training stage, architectures similar to those used in simulation, were employed. Sensors were installed to detect when the 40 W or 100 W thermal loads were active and neural networks were trained to identify plants based on these readings. Recurrent neural networks were trained using backpropagation through time as described in Section 2.2.1, and the Feedforward neural network was trained using traditional backpropagation (Wythoff, 1993) .

The training algorithms for the networks are presented in Appendices 9.1, 9.2, 9.3, and 9.4. The same considerations were made regarding the Mean Squared Error (MSE), batch size, learning rate reduction, and other hyperparameters used for training the neural networks used in the simulation. Table 3 presents the Mean Squared Error (MSE) values achieved by each model at the end of training and the number of epochs each model used to reach its MSE value for experimental dataset.

Table 3 - MSE Experimental Models

MODEL	MSE	EPOCH
SIMPLE RNN STEP10	$8.43e^{09}$	3524
SIMPLERNN BI STEP10	$3.44e^{08}$	712
LSTM BI STEP 10	$2.78e^{08}$	654
LSTM STEP 10	$1.68e^{08}$	3431
FEEDFORWARD	$5.58e^{08}$	304

3.4 Neuro-PI Control

With the plants identified, the controlled gains calculated and the neural networks trained to identify them, it is now necessary to integrate the networks with the PI controller. The PI controller has a proportional term and an integral term. The expression for the control signal is given by the sum of these two terms, as shown in Equation 27. The ideal pair of controller gains (kp and ki) are determined according to the characteristics of the first-order plant. These gains vary constantly as operating and weather conditions change.

$$\mathrm{C}(t) = k_p \cdot E(t) + k_i \cdot \sum E(t) \cdot dt \qquad 27$$

Therefore, considering the control proposal presented schematically in Figure 30, the PI controller will be tuned by deep artificial neural networks Feedforward, RNNs, LSTMs, with and without bidirectionality. The comparison between different methods allows to analyze the influence of factors such as recurrence and bidirectionality in the adaptive control process. For all simulations, there was saturation of the control signal in $\pm$ 140, and the error

signal was obtained by comparing a temperature value chosen as the setpoint and the controller output temperature.

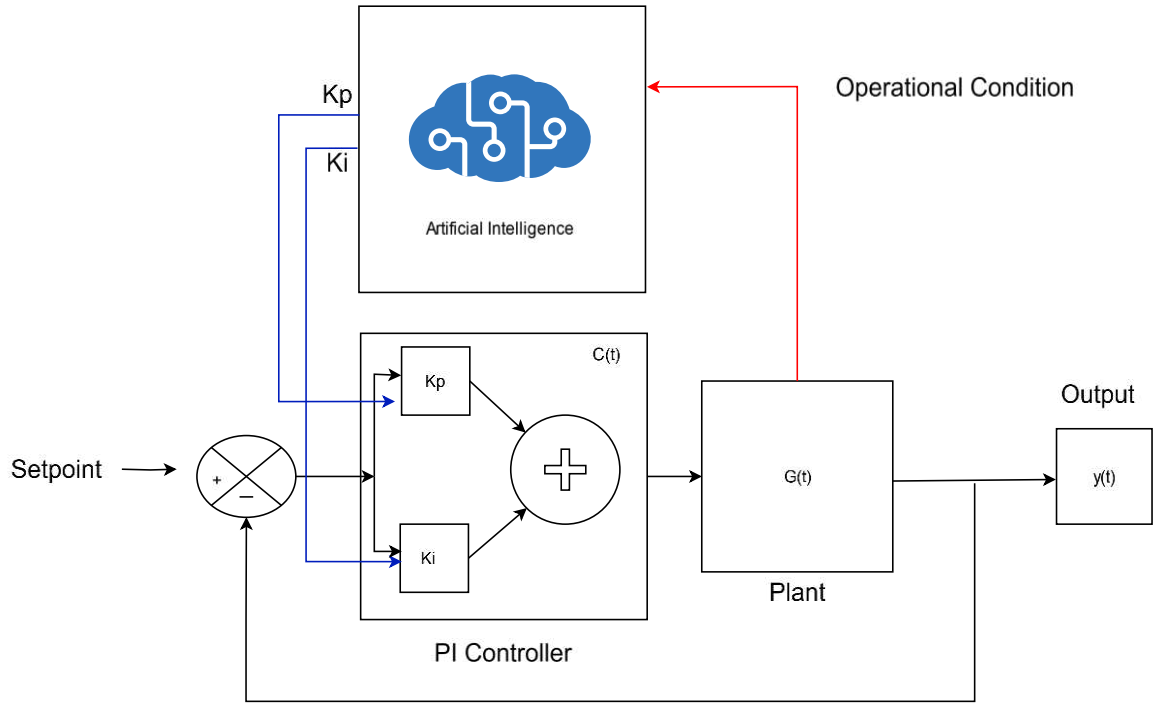


Figure 30 - Neuro PI Control

3.5 Integration of neural networks into simulation

The simulation environment was built in Python language considering a time interval of 0 to 8000 seconds. At the initial time, the local temperature was 26°C and the control algorithm used a setpoint of 16°C. For first-order systems with transport delay, the differential equation in the time domain is described by Equation 28.

$$\tau_p \cdot \frac{d_T}{dt} = -T(t) + K \cdot u(t-\theta) \quad (28)$$

Where T is the temperature, K is the static gain of the plant, τ_p is the time constant, Solving the above differential equation yields get the response in the time domain.

$$T(t) = T(0) \; if \; T(t) < T(0)$$

$$\text{Else: } T(t) = \left(e^{\frac{-(t-\theta)}{\tau}}\right) \cdot T(0) + \left(1 - e^{\frac{-(t-\theta)}{\tau}}\right) \cdot K \cdot \Delta u \quad (29)$$

Where Δu is the control signal of the PI controller and is calculated considering its proportional and integral terms. The control signal can be calculated considering the error $\left(e(t)\right)$, the accumulation of the error $(\sum e(t))$, the proportional gain $\left(k_p\right)$, the integral gain (k_i), and the simulation time step (dt) according to Equation 30.

$$\Delta u(t) = k_p \cdot e(t) + k_i \cdot \sum e(t) \cdot dt \qquad 30$$

During the simulation, the values of the proportional (k_p) and integral (k_i) gains are adjusted by the artificial neural networks based on the changes in the environment's thermal load. It was chosen to use the Python simulation environment instead of other environments, such as Simulink from MATLAB (2022) because the simulation of adaptive tuning with recurrent neural networks in MATLAB is not compatible with Machine Learning models trained with the TensorFlow libraries and the Keras API in their most recent versions. The simulation environment was developed to simulate PI control acting on first-order plants with transport delay and is presented in annex 6. To validate the Python environment, the simulation results of the control were compared with those obtained by the simulation in MATLAB. The Simulink blocks are presented in Figure 31

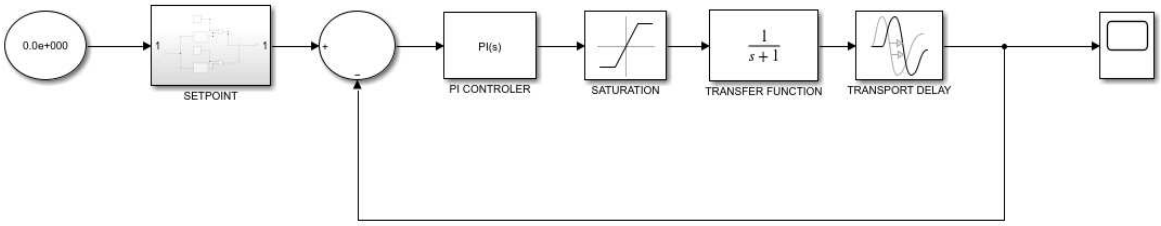


Figure 31 - PI Control Simulink Blocks

In Figure 31, the clock updates the setpoint change within the Setpoint subsystem. In the range of 0 to 2000 seconds, was simulated a setpoint of 30°C, and between 2001 and 10000 seconds, a setpoint of 35°C. The simulation results with MATLAB and Simulink, considering a first-order plant with process gain equal to 1, time constant equal to 1 and transport delay equal to 1, both in MATLAB and in the Python environment, are presented in Figure 32. It is observed that, throughout the simulation, the same results were obtained, indicating that the Python simulation environment works correctly.

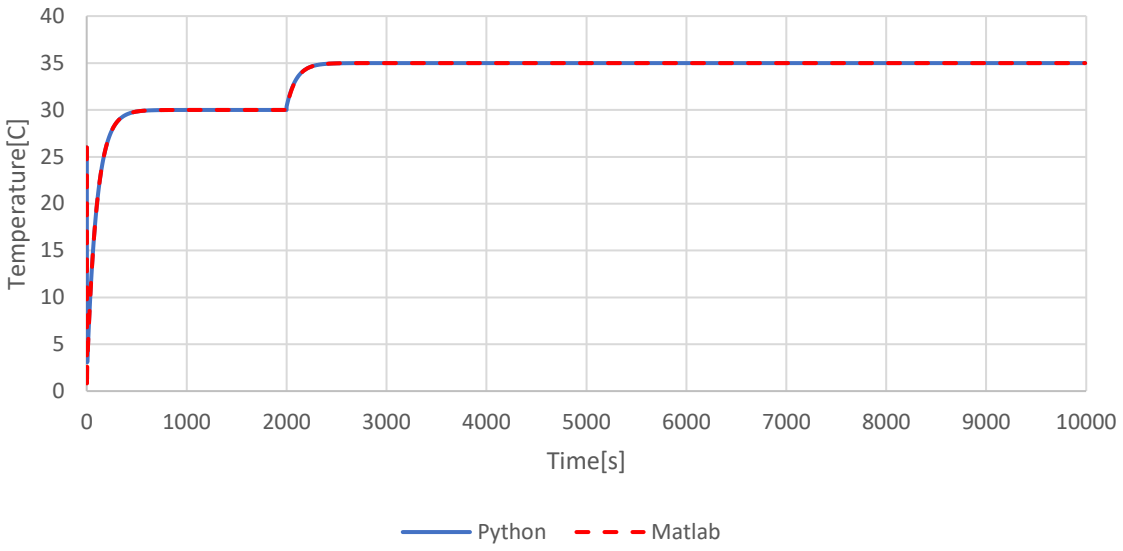


Figure 32 - MATLAB and Python Results

To simulate the Neuro-PI control using the methodology presented, it will be considered a thermal space, which receives variable and external blown air. The heat exchanger acts as an evaporator, operating in conjunction with other components of an HVAC system. The temperature of the thermal environment is constantly measured, generating the control signal (SF). Figure 33 illustrates an example of a duct where the return air comes from the conditioned environment after absorbing the thermal load. This return air mixes with fresh air and passes through the evaporator, releasing heat for the HVAC system to transfer to the external environment via the condenser unit. As the air passes through the evaporator, its temperature is reduced, and it is then ventilated into the internal environment by the fan, which is controlled by a control signal (SF).

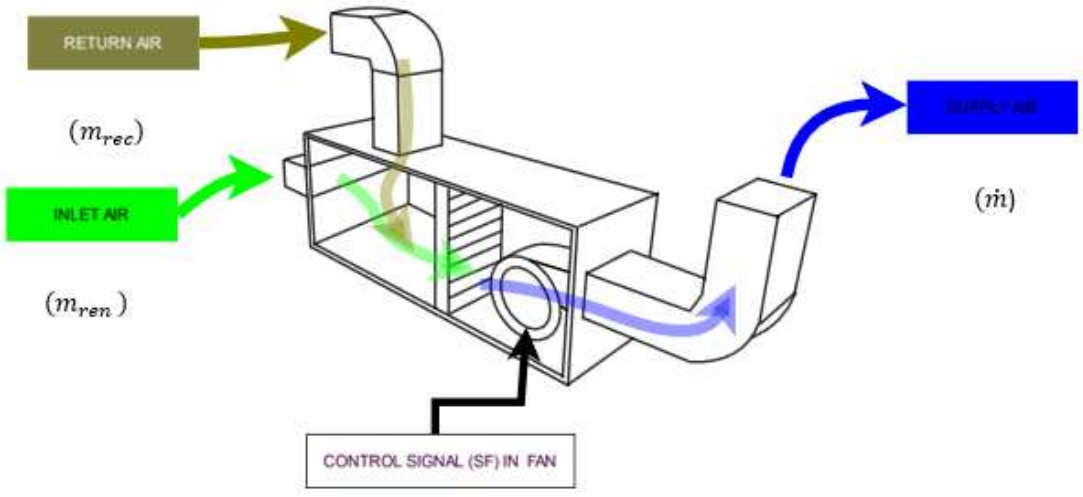


Figure 33 - HVAC Thermal Control. Source: Author

The signal (SF) controls the fan rotation and, can be calculated by Equation 27 through the implemented Neuro-PI algorithm. In simulation, it will be considered the application of only one PI controller controlling the fan rotation, for example, through the control signal (SF). The criterion used to build the air conditioned plants was initially based on the study by Yamazaki et al. (2011). From the plant proposed by Yamazaki et al. (2011), seven more plants were generated, varying the process gain, the time constant and the transport delay.

During the control task of each of these plants, the parameters were changed, increasing by 60% at approximately 3000 seconds and remaining until 6000. At the second 6000, a new parametric variation of equal magnitude occurred, and these new parameters were maintained until the end of the simulation. The changes in the plants in seconds 3000 and 6000 are associated with different mass flows of fresh and recirculated air entering the environment. The neural networks, based on these two inputs, identify the changes in the flow rates of these air flows and adjust the controller gains as the environment changes.

For each of the plants and at the three moments of the simulation – between 0 and 3000 seconds, between 3000 and 6000 seconds and between 6000 and 8000 seconds, the controller gains were calculated using the Viziole (2003) correlation, implemented in the MATLAB PID Tuner Toolbox. Controllability was verified following the methodology presented by Tan (2004) in this these. Table 4 shows the eight plants considered, with their parametric variations. In yellow in the table were highlighted the case that will be presented in the results topic.

Table 4 Plants and Controller Gains

	Time (0 – 3000s)					Time (3001 – 6000s)				
Plant	K_0	τ_0	θ_0	kp_0	ki_0	K_1	τ_1	θ_1	kp_1	ki_1
1	-0,64	18	2,4	-1,153426	-0,109155	-1,024	28,8	3,84	-0,72089	-0,04264
2	-1	18	2,4	-0,738193	-0,069859365	-1,6	28,8	3,84	-0,461371	-0,027288814
3	-2	18	2,4	-0,369097	-0,034929682	-3,2	28,8	3,84	-0,230685	-0,013644407
4	-4	18	2,4	-0,184548	-0,017464841	-6,4	28,8	3,84	-0,115343	-0,0068222
5	-0,64	22	2,4	-1,105444	-0,090616139	-1,024	35,2	3,84	-0,690903	-0,035396929
6	-0,64	26	2,4	-1,071824	-0,07741332	-1,024	41,6	3,84	-0,669890	-0,030239578
7	-0,64	30	2,4	-1,046967	-0,067548058	-1,024	48	3,84	-0,654355	-0,02638596
8	-0,64	18	8,4	-1,048027	-0,065377485	-1,024	28,8	13,4	-0,655017	-0,02553808

	Time (6001 – 8000s)				
Plant	K_2	τ_2	θ_2	kp_2	ki_2

1	-1,6384	46,08	6,144	-0,45056	-0,01666
2	-2,56	46,08	6,144	-0,288357	-0,010659693
3	-5,12	46,08	6,144	-0,144178	-0,005329847
4	-10,24	46,08	6,144	-0,072089	-0,002664923
5	-1,6384	56,32	6,144	-0,431814	-0,013826926
6	-1,6384	66,56	6,144	-0,418681	-0,011812335
7	-1,6384	76,8	6,144	-0,408972	-0,010307016
8	-1,6384	46,08	21,504	-0,409386	-0,009975813

3.6 Experimental setup

To be considered mature in the context of Industry 4.0, a company's processes must first have visibility and transparency, as illustrated in Figure 1. This means having real-time visualization tools and a clear understanding of what is happening in the plant. Only after achieving this foundation is, it possible to use advanced algorithms for forecasting (step 5 of Industry 4.0) and task automation (step 6 of Industry 4.0). To reach this level of maturity in the refrigeration industry, it is necessary to develop a computational infrastructure through the development of cyber-physical software.

Most of the literature addressing HVAC system control using artificial intelligence techniques focuses solely on simulation, validating and analyzing the use of machine learning models numerically (Zou et al., 2020). The difficulty in applying deep learning algorithms to adaptive PI control lies in the complexity of developing the computational system that enables the technique to be operationalized. This chapter will describe the technologies used in developing the software for thermal control of an environment climatized by an HVAC system.

Once the control project with machine learning software is complete, experiments will be conducted controlling the temperature of an environment at four setpoints. The experiment aims to achieve adaptive PI algorithm control using different machine learning algorithms, namely: Deep Feedforward, Depp recurrent neural networks, Deep bidirectional recurrent neural networks, long short-term memory networks (LSTM), and bidirectional LSTM. The temperature of the climatized environment will be controlled at setpoints of 21, 22, 23, and 24°C. In the first 500 seconds, the thermal load is 40 W, and from the 501st second onward, the thermal load is 100 W. The goal is to evaluate whether adaptive tuning with these different types of algorithms offers advantages over others. Additionally, control without gain adaptation, ON/OFF control and deep recurrent neural networks directly control was performed for comparison with the results.

The refrigeration circuit consists of a condenser, an evaporator, a compressor, and a thermostatic expansion valve. A filter dryer was installed at the expansion valve's inlet, and type T thermocouples with ±0.5ºC uncertainty were used for temperature acquisition in conjunction with a data acquisition system that uses the NI-USB 9162 board from National Instruments and the LabVIEW control interface. The evaporator climatizes an environment with internal thermal load provided by two lamps that can be on or off. One lamp has 40 W of power, and the other has 100 W.

The control is exercised on the evaporator fan, which is rotated by a three-phase motor controlled by a frequency inverter from manufacturer CFW 08. The frequency inverter is controlled by an analog signal from an amplifier circuit, which in turn receives the signal from the Raspberry Pi 4 Model B, which receives input from a neural network algorithm running on AWS Cloud and temperature data from LabVIEW via the FTP file transfer protocol. The Figure 34 shows the experimental setup.

List of Components			
1	Condenser and Fan	7	Frequency Inversor
2	Expansion Valve	8	Amplifier Circuit
3	Three-Phase Fan	9	Raspberry pi 4
4	Compressor	10	DC Power Suply
5	Evaporator	11	Data acquisition
6	Thermal Zone isolated	12	Dashboard

Figure 34 - Experimental Setup

The compressor used is an Embraco brand with 1 HP power and is hermetic, which allows the refrigerant fluid to expand inside, avoiding fluid compression in the liquid phase. The condenser, also from Embraco, has a heat dissipation capacity of 1 HP. The evaporator, manufactured by Elgin, has a capacity of ¼ HP. The expansion valve, made by Danfoss, was chosen with an orifice size of three. Bourdon-type pressure gauges were installed between the inlet and outlet of the expansion valve. T-type thermocouples, calibrated and with an approximate uncertainty of 0.5 ºC, were installed to monitor when one or another thermal load is activated, to monitor the temperature at a central point of the environment, and the temperatures of the refrigerant fluids at the compressor inlet, evaporator inlet, environment, and condenser outlet. The calibration process for the thermocouples is documented in the master's dissertation of Ramon de Paoli Mendes (2020).

There is ventilation for both the condenser and the evaporator, with the evaporator located inside the environment to be cooled. The fan blade is driven by a three-phase motor from the manufacturer WEG. The cooled environment has an upper duct that allows air recirculation, causing the air to be heated by the thermal loads, rise, enter the duct, and be directed back to the evaporator, where it will reject the heat to the HVAC system. The greater the ventilation provided by the fan blade, the greater the air recirculation and consequent heat exchange, reducing the temperature in the environment.

The controlled element in the HVAC circuit is the rotation of the three-phase motor. This type of motor changes the shaft rotation based on the network frequency. For this, the CFW08 frequency inverter is used, which changes the frequency of the three-phase network manually and also remotely via an electronic signal that arrives at its port 6. However, the signal from the GPIO12 port of the Raspberry Pi 4 Model B via PWM does not have enough voltage to control the frequency inverter. Therefore, an operational amplifier is needed between the output of the Raspberry Pi and the input of the frequency inverter. The LM358 amplifier, a non-

inverting amplifier that amplifies the input signal based on two resistors connected to it, was used. The Equation 31 show the amplifier in function electrical resistance.

$$V_{out} = V_{in} \cdot \left(1 + \frac{R_2}{R_1}\right) \quad 31$$

R_1 and R_2 are electrical resistances and measuring 150 and 300 oms in this work. Figure 35 shows the frequency inverter in the remote access configuration and being controlled by the signal arriving at port 6 from the LM358 operational amplifier. It is necessary to configure the range within which the frequency will be adjusted inside the frequency inverter. In this work, the frequency range of the inverter was set between 0 and 60 Hz, with 60 Hz being the standard frequency of electrical networks in Brazil.

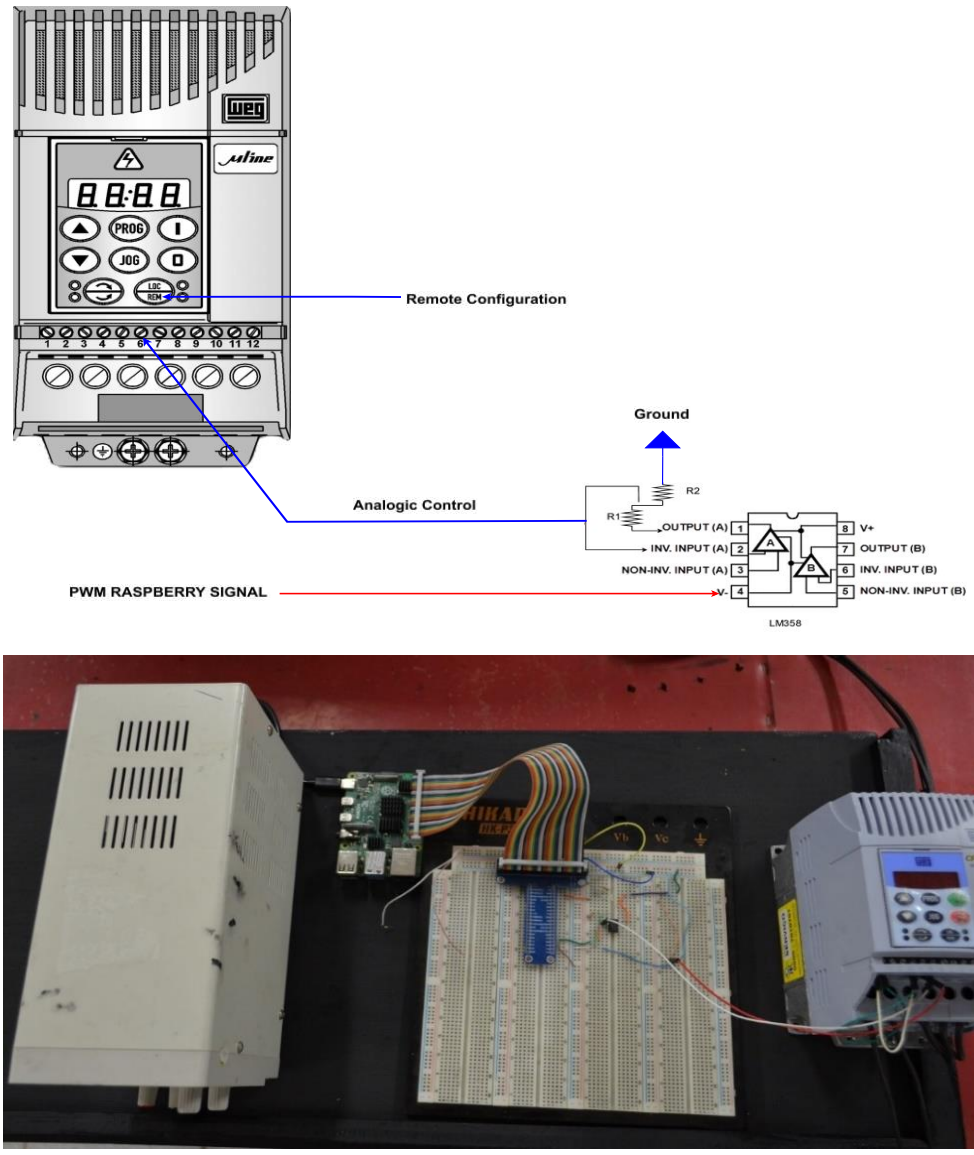


Figure 35 - Frequency Inversor and AmpOP

The PWM signal from the Raspberry Pi is calculated by the PI algorithm as per Equation 30 and then multiplied by 100 to adjust the magnitude to the 0-100% range, which corresponds to the PWM duty cycle range. The controller gains in this process are tuned via HTTP communication with the AWS cloud platform, where the neural networks are processed. The code for the PI algorithm integrated with HTTP communication for adaptive tuning of the controller is presented in Appendix 7. In section 3.7, the development of the software that enabled HTTP communication, as well as the LABVIEW code for data acquisition and transmission, will be described.

3.7 Project software machine learning control

The enable control using artificial neural networks, it was necessary to develop the entire computational infrastructure to operationalize it. The software can be divided into three parts: a backend where a relational database was built using Postgres (Postgresql 2024) to store the physical data measured by the sensors - Figure 36 . It also stores the controller gains to be consumed by the microcontroller, in this case, the Raspberry Pi 4 Model B.

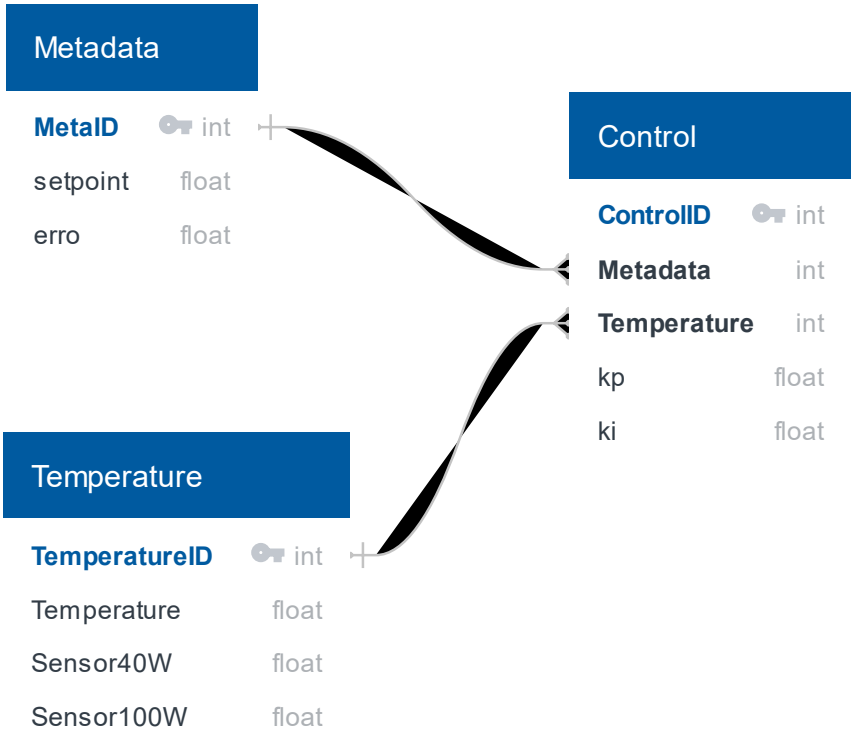


Figure 36 - Dataset Model

In Figure 36, the Metadata table is responsible for storing the setpoint data and error values. The Temperature table stores the measured temperature values from the environment

and the 40 W and 100 W sensors. The control table stores the controller gain data, which are processed by the neural networks and transmitted to the table via HTTP Post communication. These data are then available to be consumed via HTTP GET communication by the Raspberry Pi.

The user interface (Frontend) was built using HTML for text markup, JavaScript for making HTTP requests, and the Bootstrap framework to ensure a pleasant design. The communication protocol with the database chosen was HTTP. Communication between the Frontend and Backend, as well as HTTP communication between the microcontroller and the database, was made possible through the construction of an API (Application Programming Interface). The Figure 37 presented the frontend of the software.

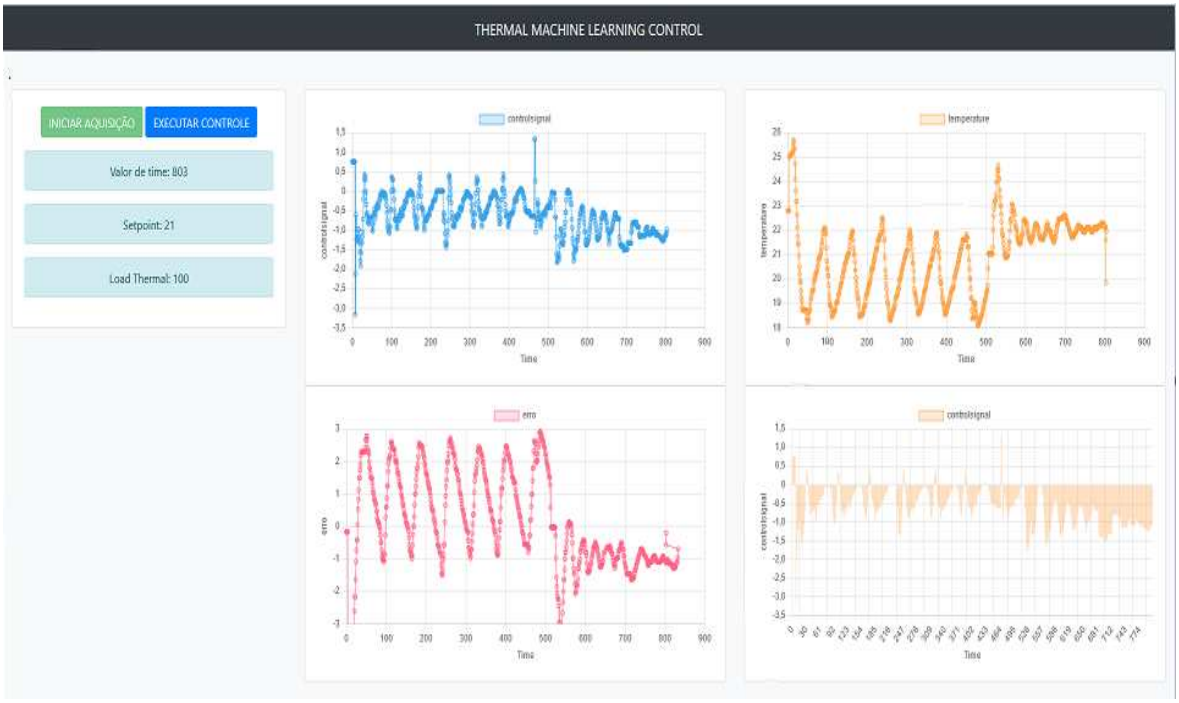


Figure 37 - Dashboard Frontend software

The API was built using FASTAPI (FastAPI 2024) with the SQL Alchemy library to facilitate access to the Postgres database using asynchronous programming. The software architecture is presented on the left side, and the API details are presented on the right side in Figure 38.

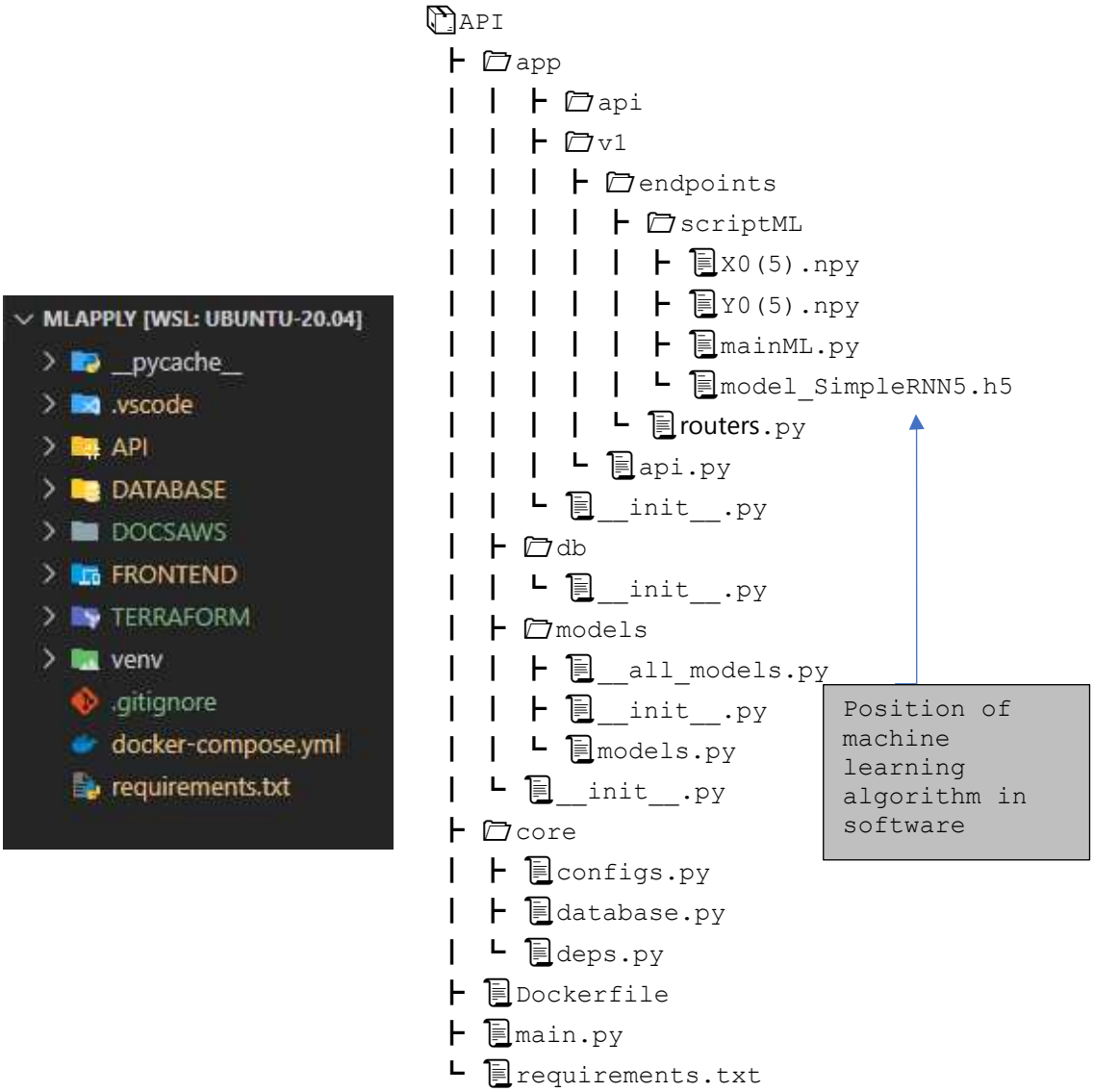


Figure 38 - Software Structure

Figure 38 too shows the positioning of the Machine Learning algorithms within the API. The file model_SimpleRNN5.h5 is the model trained with TensorFlow and Keras, ready to provide the controller gains based on the system's physical inputs. The file mainML.py is the Python script that, through HTTP requests with the database, provides the necessary data for the model's input and also transfers the model's output data to the database. The files X0(5). npy and Y0(5). npy are NumPy arrays used to denormalize the model. Figure 39 exemplifies the Python algorithm that consumes physical data from the database and generates controller gains to be stored in it. HTTP GET 1 and HTTP GET 2 will be the temperature data read by the 40 W and 100 W sensors and HTTP POST KP and HTTP POST KI are the proportional and integrative gain data that will be stored in the database and consumed by the microcontroller.

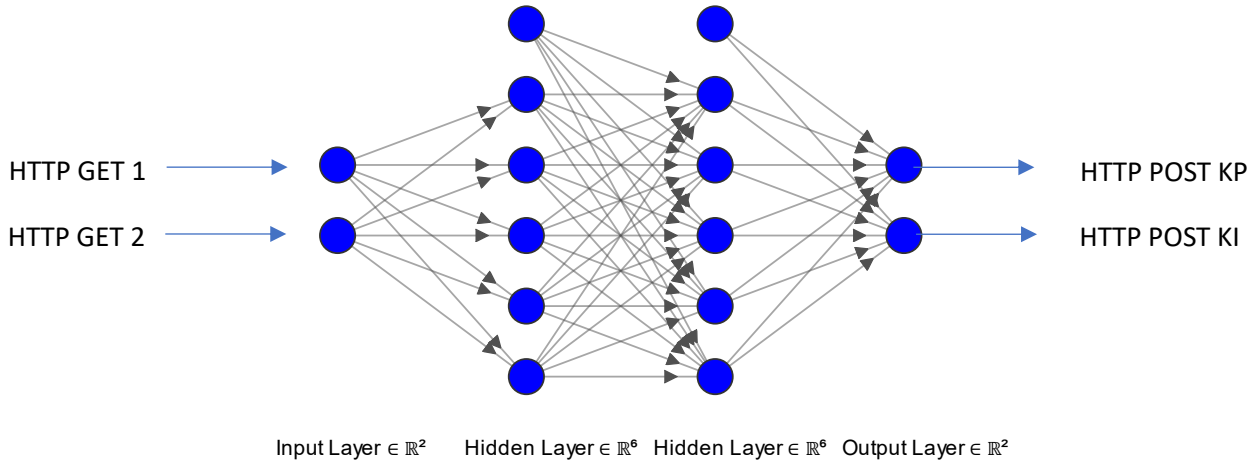


Figure 39 - Example of a Python algorithm that consumes an ML model. Source: Author

On the left side of Figure 38, the __pycache__ folder is used by Python to store compiled bytecode files. These files have the .pyc or .pyo extension and are pre-compiled versions of the .py source code files. The main advantage of __pycache__ is to improve program performance by allowing Python to load bytecode directly, rather than recompiling the source code every time the program runs. This is especially useful for large projects where compiling each source code file can be time-consuming.

The .vscode folder is created by Visual Studio Code (VS Code) to store project-specific settings and editor customizations. The DATABASE folder contains the relational database file and the Dockerfile, which will be discussed later. The DOCSAWS folder holds the access keys that enable SSH communication with the AWS cloud. The FRONTEND folder contains the code for the graphical monitoring interface that operates over the internet and is accessible to the user. The TERRAFORM folder includes the terraform.tf file, which allows the creation of the cloud infrastructure on which the software is implemented. The venv folder is the Python virtual environment created to isolate the project from other Python components installed on the machine.

The .gitignore file is necessary to prevent unnecessary files from being stored in the computer's memory through version control processes with GIT. The docker-compose.yml file connects the other Dockerfile files within the FRONTEND, API, and DATABASE folders, enabling the construction of all Docker containers simultaneously and on the same network. In other words, the docker-compose.yml file facilitates the organization and management of Docker containers.

Regarding Docker containers Docker (Docker 2024), they were used to ensure the operational integrity of the software and its functionality on any computer system. Using

software within containers facilitates portability between operating systems and its use in cloud computing systems such as AWS (Amazon Web Services), Microsoft Azure, or Google Cloud. In this work, AWS cloud processing was chosen using an EC2 (Elastic Compute Cloud) instance, but any other platform could be used since the API's design allows for this portability. Figure 40 shows the Docker containers for the software components in the EC2 AWS instance. The API communicates with the database and the frontend, receiving HTTP requests from the internet. The database communicates only with the API, while the frontend interacts with the API and provides the visual interface to the user.

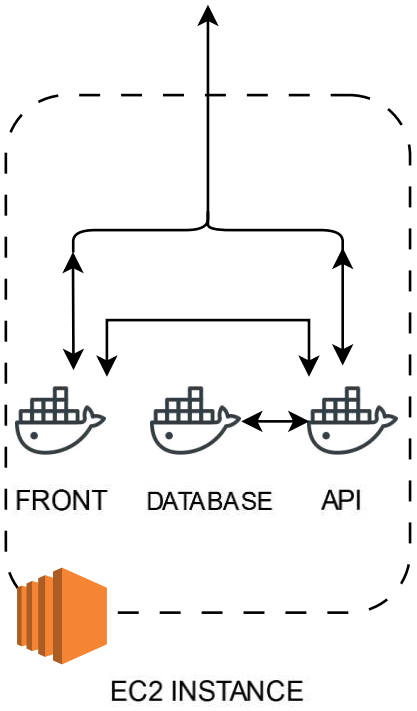


Figure 40 - Software in EC2 Instance

The neural network processing could also be done directly within the Raspberry Pi 4, thus avoiding the cloud project and cloud processing. However, the technology that would allow its use on the Raspberry Pi, TensorFlow Lite (Org, Tensorflow), discretizes the neural network model so that it is compatible with the microcontroller's processing capacity. This reduces precision and therefore—in this work—cloud processing was chosen to ensure the good accuracy of the machine learning models used.

Temperature readings are sent to the Raspberry Pi 4 directly from LabVIEW via FTP communication, while simultaneously being stored locally in an Excel file for later analysis. Temperature data is transferred to the Raspberry Pi 4 almost instantaneously, with transmission time being less than 1 second. Figure 41 shows the LabVIEW code used. The top block within the main loop establishes FTP communication with the Raspberry Pi and sends the read

temperature data. The temperature data is read through the DAQ Assistant and stored in two Excel files via the intermediate loop in Figure 41: one file is for backup, and the other contains the data that is sent to the Raspberry Pi. To ensure the transmitted Excel file is not too large and does not affect transmission time, the lower loop deletes previous data from the file being sent to the Raspberry Pi, ensuring only instantaneous data reaches the microcontroller.

Figure 42 shows the LabVIEW user interface. The first button on the left, when pressed, allows for sending data to the Raspberry Pi 4. The middle button, when pressed, starts data acquisition. The graph monitors the environment temperature over time, and the lower fields must be filled with the Raspberry Pi data and also with the path of the local Excel file to be transmitted, as well as the path where it will be received within the microcontroller.

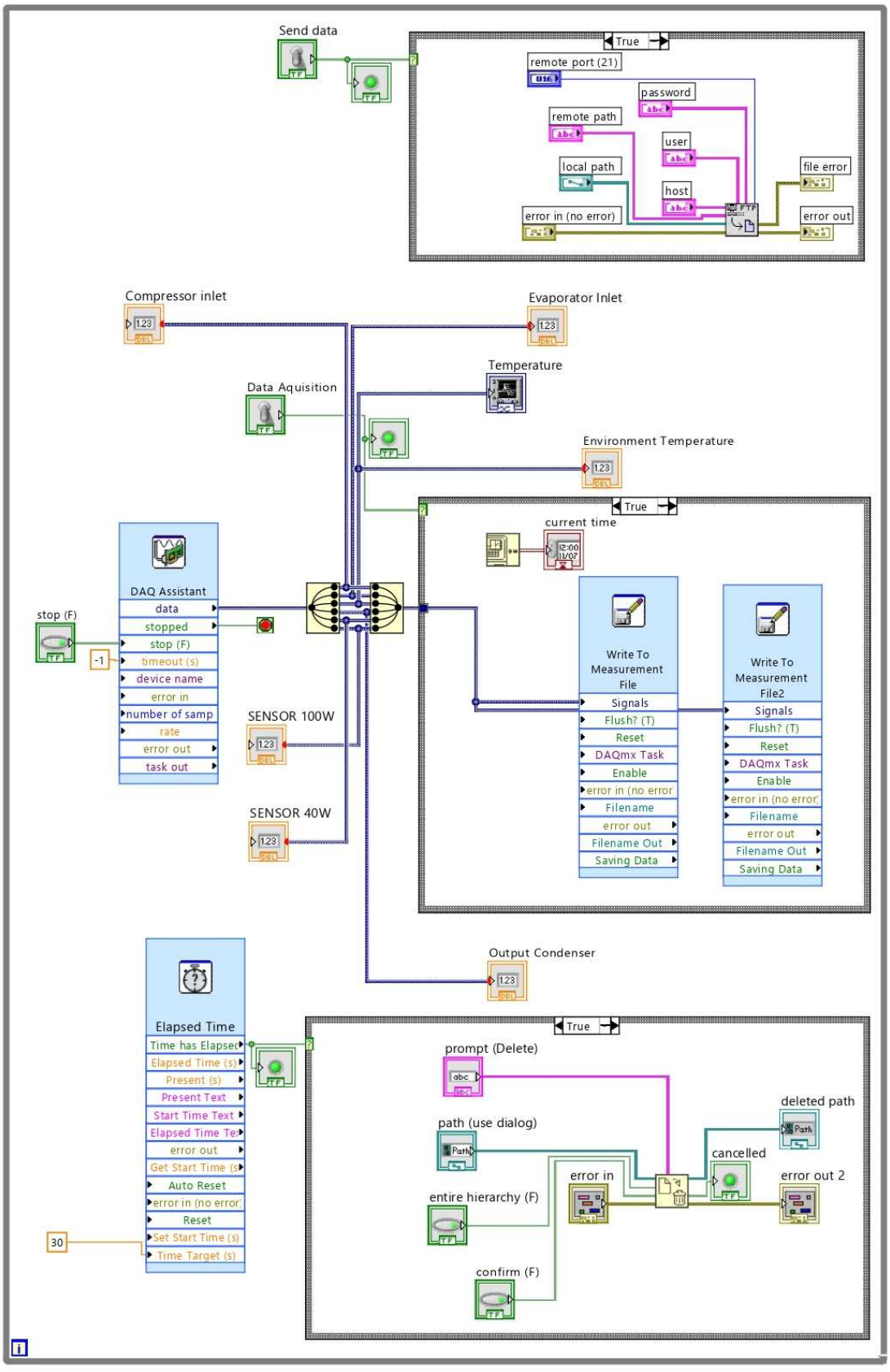


Figure 41 - Labview Aquisition Data

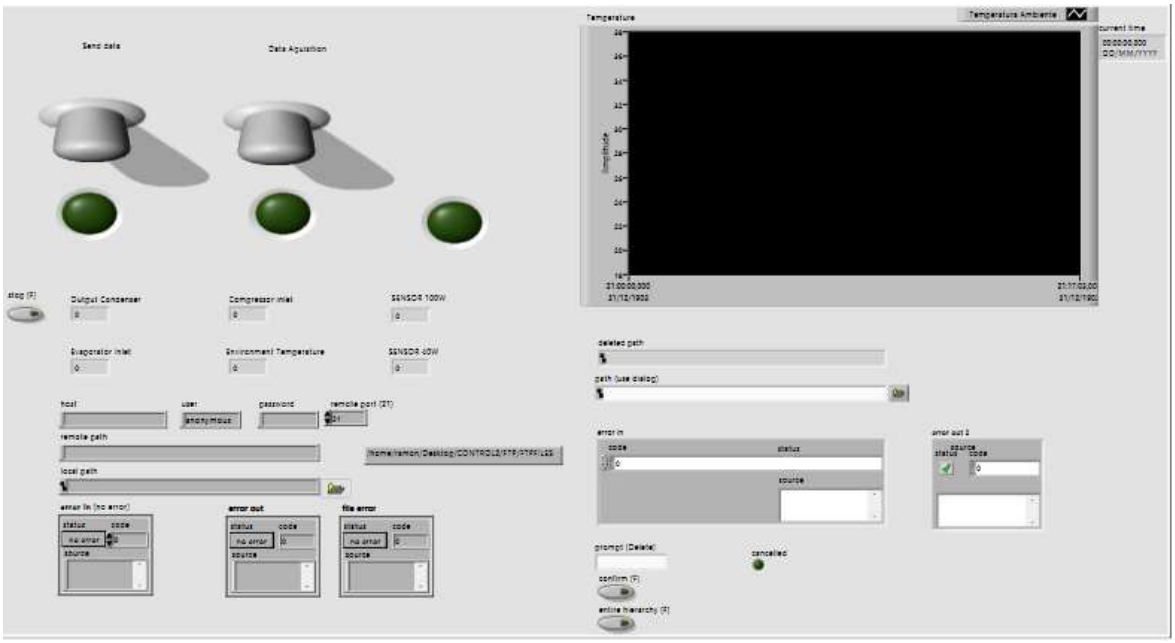


Figure 42 - LabVIEW Graphical Interface

To ensure the software operates safely in processing neural networks, it was necessary to build the infrastructure in the cloud. Figure 43 presents the infrastructure project built on the AWS cloud. The VPC (Virtual Private Cloud) is where the entire infrastructure is constructed within the platform. The Internet Gateway is the portal that provides access to the internet. Two availability zones ensure that if a problem occurs with one, the other will continue operating, thus preventing interruption of control. The Load Balancer ensures that if an overload occurs in one availability zone, data traffic is redirected to another. Both the NACL (Network Access Control List) and the Security Groups act as firewalls, allowing only reliable traffic. The EC2 is the computer itself, provisioned on AWS and with an installed operating system. Inside the EC2, Docker containers were allocated with the frontend, backend, and API projects presented in Figure 41. The main commands for using Docker containers in software, as well as for connecting to and building the AWS infrastructure, are presented in Annex 9.8.

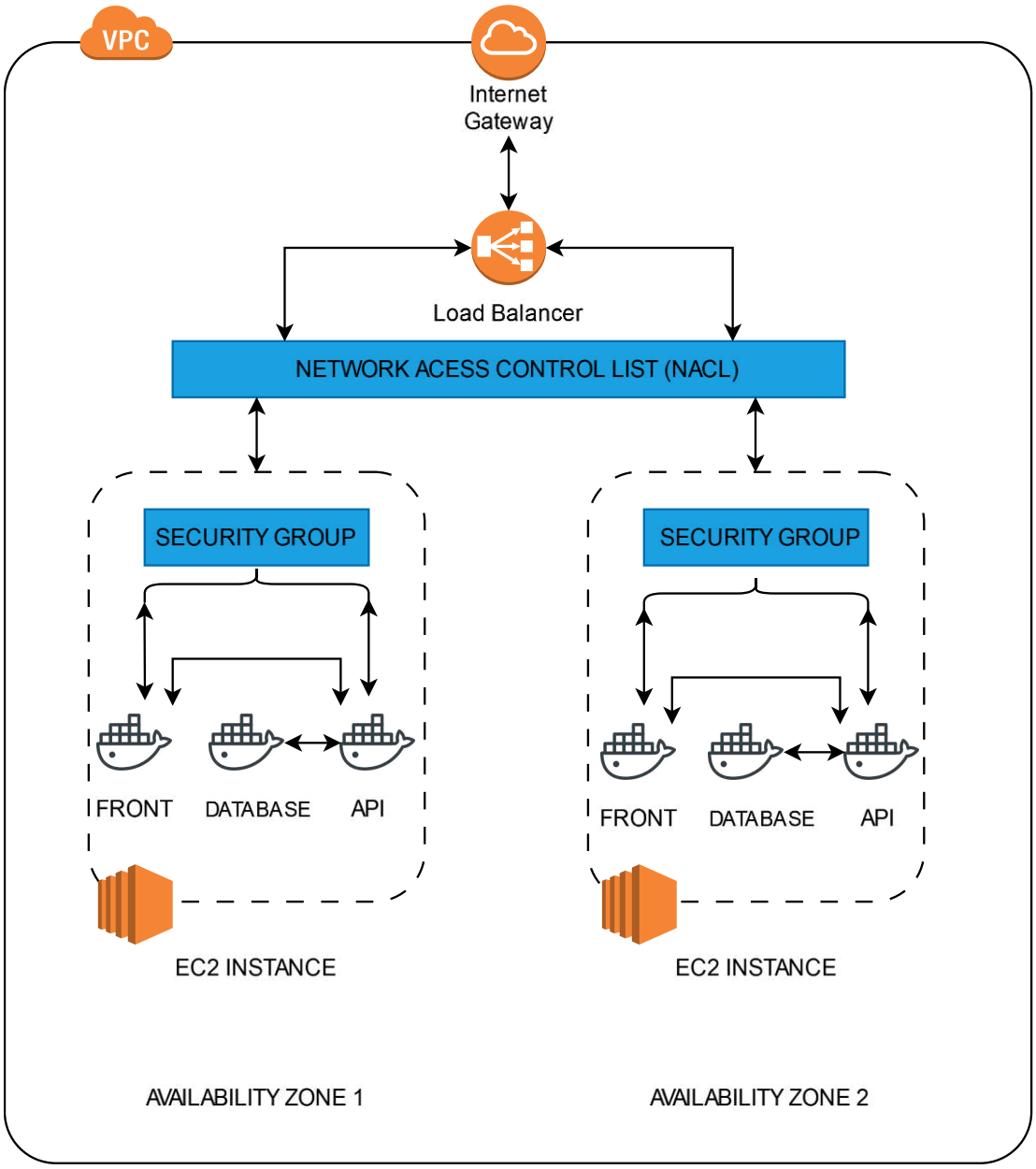


Figure 43 - Project AWS Cloud

The microcontroller used, the Raspberry Pi 4 Model B, was chosen due to its processing capacity and high wireless connectivity. The Raspberry receives the controller gains, already processed by the neural network operating in the cloud, and inputs data into it. Additionally, it receives data via the FTP protocol from the data acquisition system that uses LabVIEW software. The Raspberry Pi 4 Model B has two PWM signal ports (GPIO 12 and GPIO 13), whose signal can be modulated by adjusting the duty cycle.

Once the data is received on the Raspberry Pi, it is used to calculate the control signal through the PI algorithm and adjust the PWM on GPIO 12. The PWM signal at the microcontroller output is low, and to be used in the frequency inverter, it needs to be amplified.

For this, an LM358 non-inverting operational amplifier was used, which amplifies the signal at the Raspberry Pi output based on the eletrical resistances (R1 and R2) connected to it com.

The amplified signal reaches the CFW08 frequency inverter, which can be controlled remotely. The inverter was configured to operate within a frequency range of 0 to 60 Hz upon receiving the control signal that operates in the range of 0 to 100%. Upon receiving the control signal, the inverter adjusts the output frequency, which in turn controls the fan's rotation in the HVAC system's evaporator. The fan with a higher rotation increases air circulation within the climatized environment, thereby increasing the heat exchange coefficient and consequently reducing the temperature, maintaining it at a specific setpoint.

The complete project for applying machine learning algorithms to temperature control in an environment conditioned by an HVAC system is presented in the Figure 44.

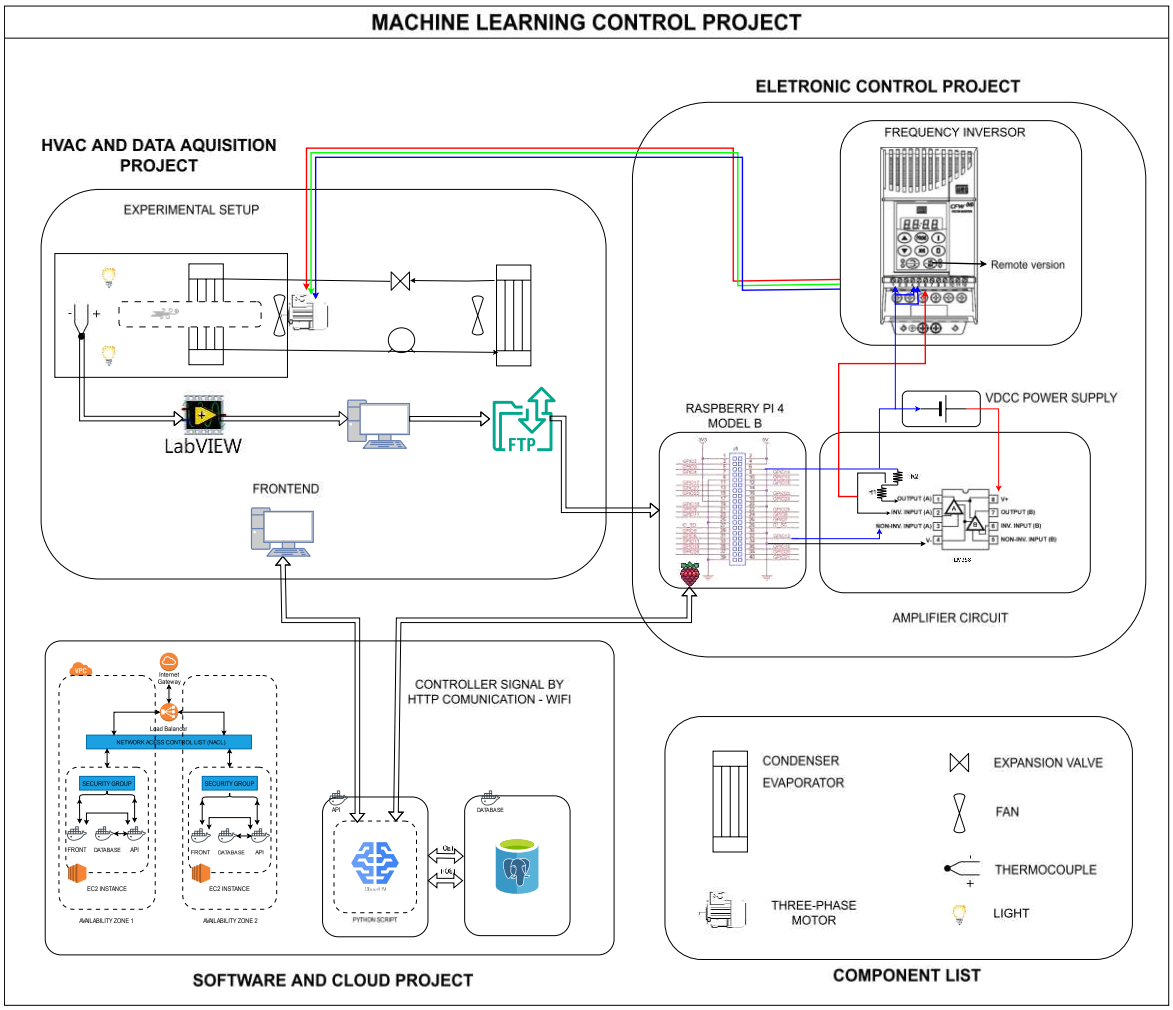


Figure 44 - Complete project Machine Learning control

3.8 Closure for methodology

In this chapter, the methodology of the work was discussed. Item 3.1 presented how the plants to be controlled were obtained, both for the simulations in the Python environment and for the experiment. In the simulations, the plants were derived from the one experimentally obtained by Yamazaki (2011), and the controller gains were tuned in the simulation based on the mass flow rates of air for renewal and recirculation. The two experimental plants were obtained through graphical analysis of the step response for each thermal load (40 W and 100 W).

With the plants in hand, item 3.2 presents the methodology for calculating the PI controller gains for each plant and how to verify the stability of these gains concerning control. Once stability was ensured, item 3.3 demonstrates how neural networks were built to identify the gains, with the simulation accounting for changes in air mass flow rates and the experiment considering the incident thermal load. It explains how the architectures were obtained and how the models were monitored to select the most appropriate one.

Item 3.4 demonstrates how neural networks interact with the PI algorithm. Item 3.5 details how the simulation environment was built and validated. With the simulation environment built and validated, item 3.5 further details how the simulation was conducted. Table 4 presents all the simulated plants. Essentially, 7 plants were obtained with variations of Yamazaki's plant, and in each of them, variations in the mass flow rates of recirculating and renewing air were simulated at seconds 3000 and 6000, causing the parameters to vary by 60%. The neural networks identified these changes in flow rates and predicted the controller gains for the new plants. The architecture shown in Figure 39 is similar to that used for the simulation, with two inputs associated with air flows and two outputs associated with controller gains.

Item 3.6 presents the experimental setup and how the microcontroller interacts with the amplifier and frequency inverter to receive the signal from the cloud algorithm and control the three-phase fan. Item 3.7 presents the entire software and cloud infrastructure design that allows the experimental operation of the HVAC system control using Deep Learning algorithms.

4 RESULTS AND ANALYSIS

Regarding the simulation control, during the training stages, the networks with bidirectionality presented a smaller number of epochs compared to the same networks without bidirectionality, indicating a greater learning capacity with the use of bidirectionality. Figure 45 shows the accumulated training epochs considering all trained networks. Eight plants were analyzed, with their parametric variations according to Table 4, and Deep Feedforward, SimpleRNN (Deep recurrent algorithm without bidirectionality), SimpleRNN (Deep recurrent algorithm with bidirectionality), LSTM and bidirectional LSTM networks were trained. This accumulation refers to 40 simulations of the Table 4.

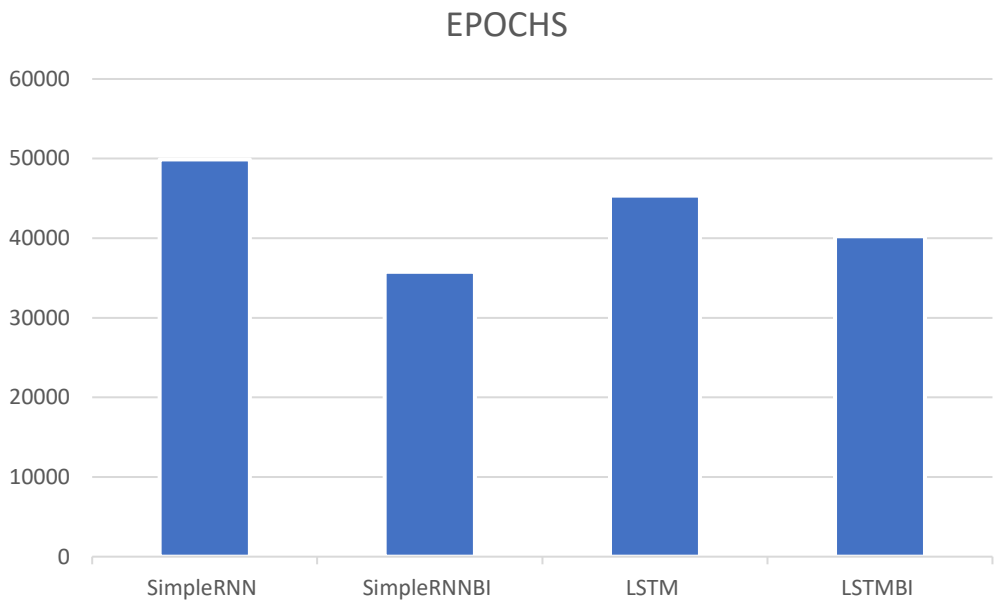


Figure 45 - Accumulation of epochs

Regarding temperature control at the setpoint of 16ºC, a parametric variation was carried out at approximately 3000 and 6000 seconds, according to Table 4. All simulations, for the 8 plants at the 3 simulation times and following the methodology proposed, showed the same behavior. All simulation results are presented in the Figure 46. In Figure 46, all the graphs that reduce the temperature from the 1990th second onwards are those that were tuned in the PI controller with recurrent algorithms, while those that reduce the temperature at the exact 3000th second are the ones tuned in the PI controller by feedforward neural networks. It is noticeable that the main difference between tuning with feedforward and recurrent networks is that recurrent neural networks anticipate the increase in ventilation that will occur and adjust the controller gains within the time step with which they were trained.

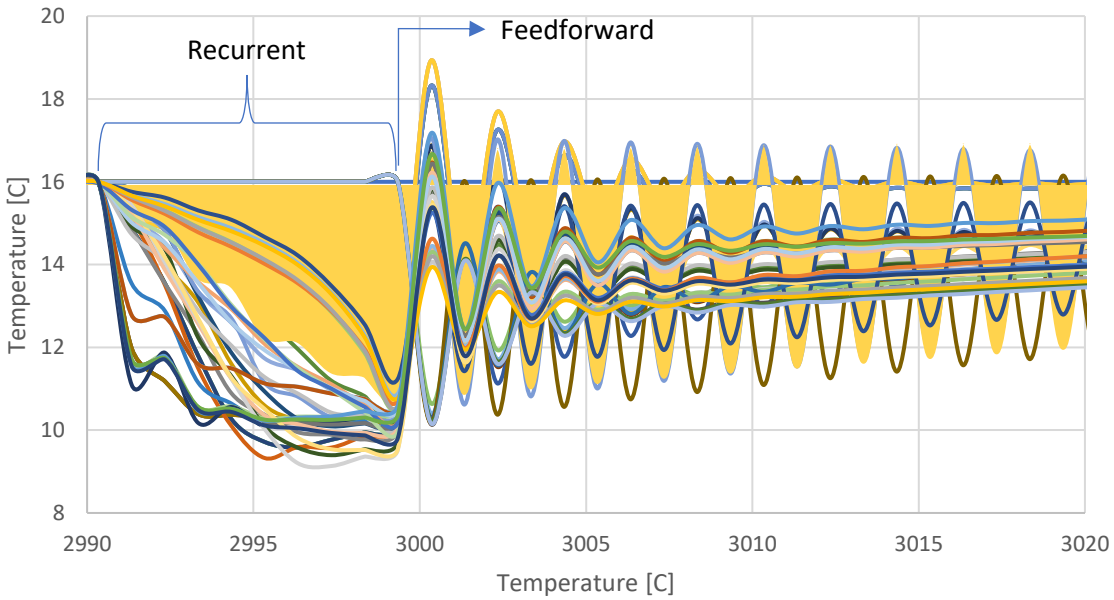


Figure 46 - Simulation results between 2990 and 3020 seconds

To facilitate the visualization of the results, Figure 47 presents those associated with Plant 6, highlighted in yellow in Table 4. In Figure 47a, the control simulations are shown over 8000 seconds, highlighting the parametric changes, ventilation change, around 3000 and 6000 seconds. In turn, Figure 47b highlights the interval between seconds 2990 and 3010, revealing the temperature variations according to the neural network used for tuning. Figure 47c is similar to Figure 47b, but presents the data between seconds 5990 and 6010 and emphasizes the time step used in the recurrent algorithms.

When analyzing Figure 47b, it is noticeable that the adaptive tuning by the feedforward network causes abrupt changes in temperature at the times of parametric changes. In contrast, recurrent neural networks start the transition in advance, following the time step established during training, which was ten seconds. This characteristic is attributed to the memory effect of recurrent networks.

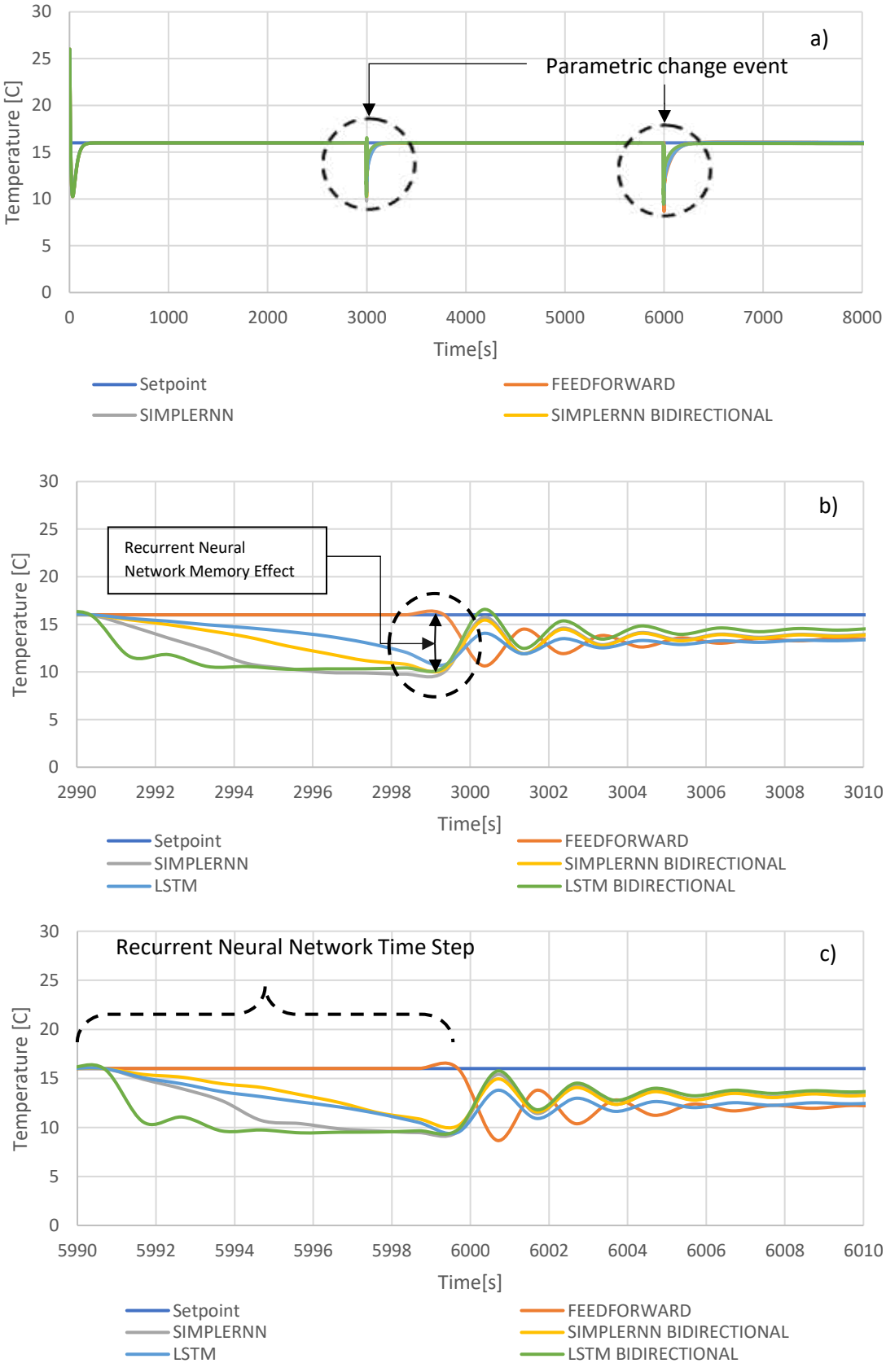


Figure 47 - Control simulation for plant 6. A) 0 - 8000s, b) 2990 - 3010 s, c)5990 - 6010 s

Recurrent neural networks start to change the temperature in a time equal to that used as the time step during the training phase, as can be seen in Figure 47c. However, when the post-parametric change event time is reached, the recurrent adaptive tuning tends to change the temperature to the desired setpoint. It is important to note that regardless of the model used, all networks were able to control the plant even in function of the parametric changes that occurred. Considering plant 6 of the Table 4 and if adaptive tuning were not used, the controller would not be able to control it. As can be seen in Figure 48.

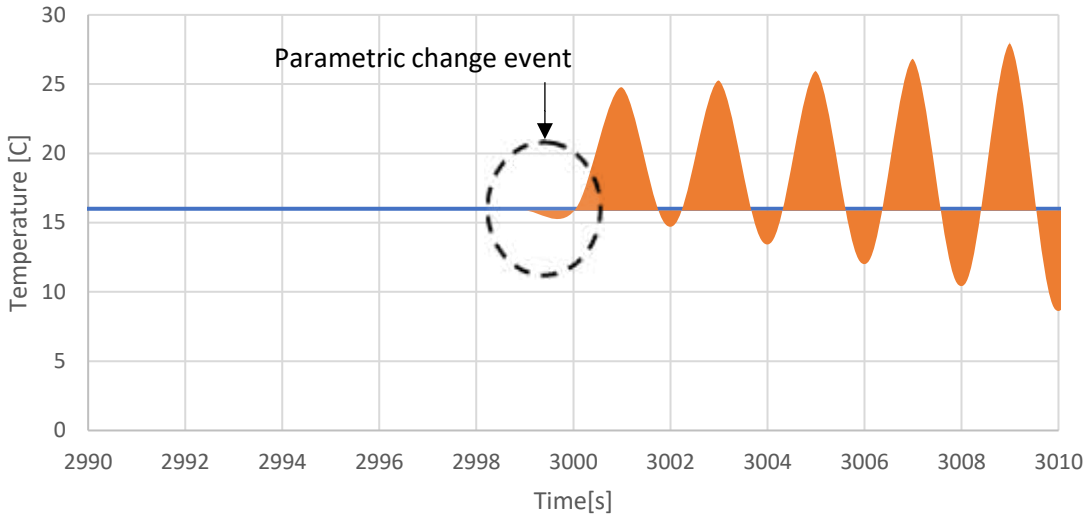


Figure 48 - Plant without adaptive tuning

The Figure 49 and Figure 50 shows the proportional and integrative gains variations. Controller gains tuned by recurrent neural networks begin the transition between seconds 2990 and 3000. Because the recurrent networks were trained with a ten-second time step, they begin the transition ten seconds in advance, whereas the deep feedforward neural networks adjust the controller gains almost immediately upon the event change. This characteristic is important as it helps the controllers tune the gains according to the trends perceived by the sensors and not based on just a single piece of information that defines a ventilation level.

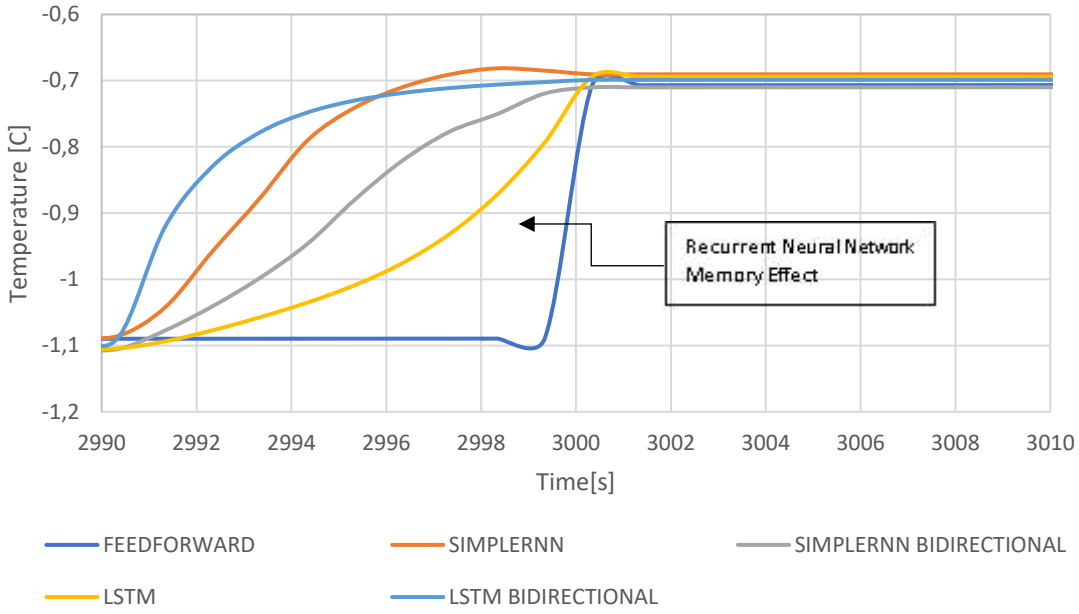


Figure 49 - Variation of proportional gain for the first parametric change event (2990 to 3010 s) of plant 6

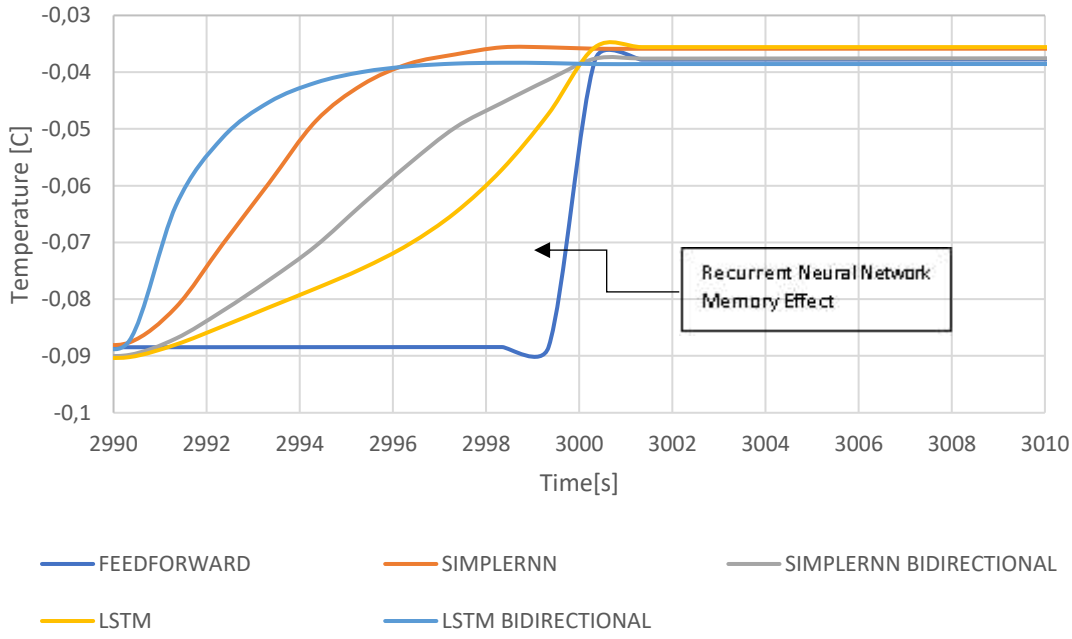


Figure 50 - Variation of the integrative gain for the first parametric change event (2990 to 3010 s) of plant 6

The same behavior observed in Figure 49 and Figure 50 for changes in the proportional and integral gains, respectively, was observed for all other simulations. While the controller gain changes almost like a step response for adaptive tuning with feedforward neural networks, for recurrent neural networks the change is gradual and distributed within the time step used during the training phase.

Regarding experimental results, to control the temperature at a specific setpoint, the energy that would naturally heat the environment is transferred to the HVAC system, which then transfers the energy to another environment. Sensors were installed at the condenser outlet, the evaporator inlet, and other points on the experimental bench. Figure 51 shows the temperature readings at the condenser, evaporator, and within the conditioned environment. The change in thermal load, from 40 W to 100 W, occurred at second 920, and from that point, it is observed that the ambient temperature remained controlled at the setpoint of 22ºC, while temperatures at the condenser and evaporator increased.

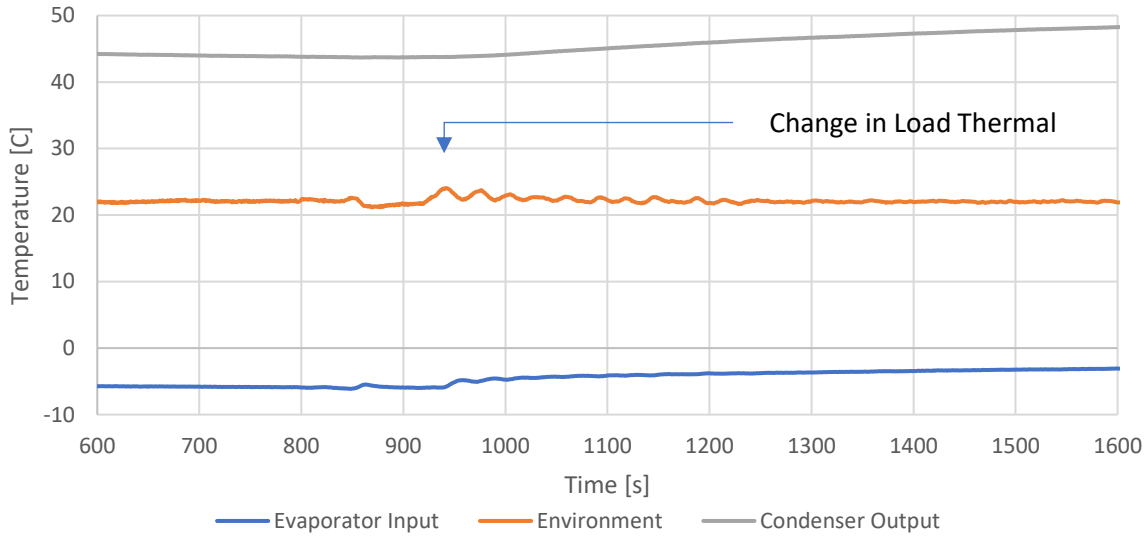


Figure 51- Experimental Temperature Measurement

The thermal environment was conditioned using PI control adaptively tuned by deep feedforward and deep recurrent neural networks and LSTM with and without bidirectionality. Additionally, ON/OFF control, non-adaptive tuned PI control and directly control by Deep Recurrent neural network were tested for comparative purposes. The temperature was controlled at setpoints of 21, 22, 23, and 24°C, and the plant change occurred at the 500 second in all experiments. Figure 53 presents the overall control results for all setpoints. Figure 54 shows the same results but focuses on seconds 450 to 650, with the plant change occurring at second 500. The ON/OFF control showed the worst results in all experiments, and its results were shows individually in Figure 52.

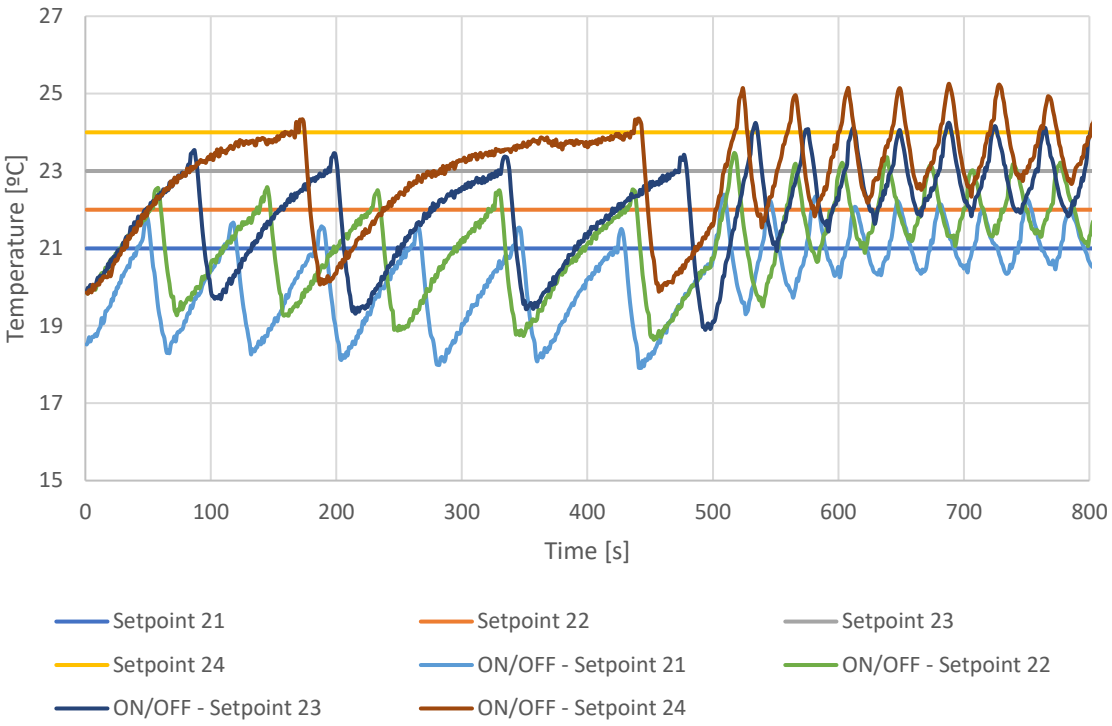
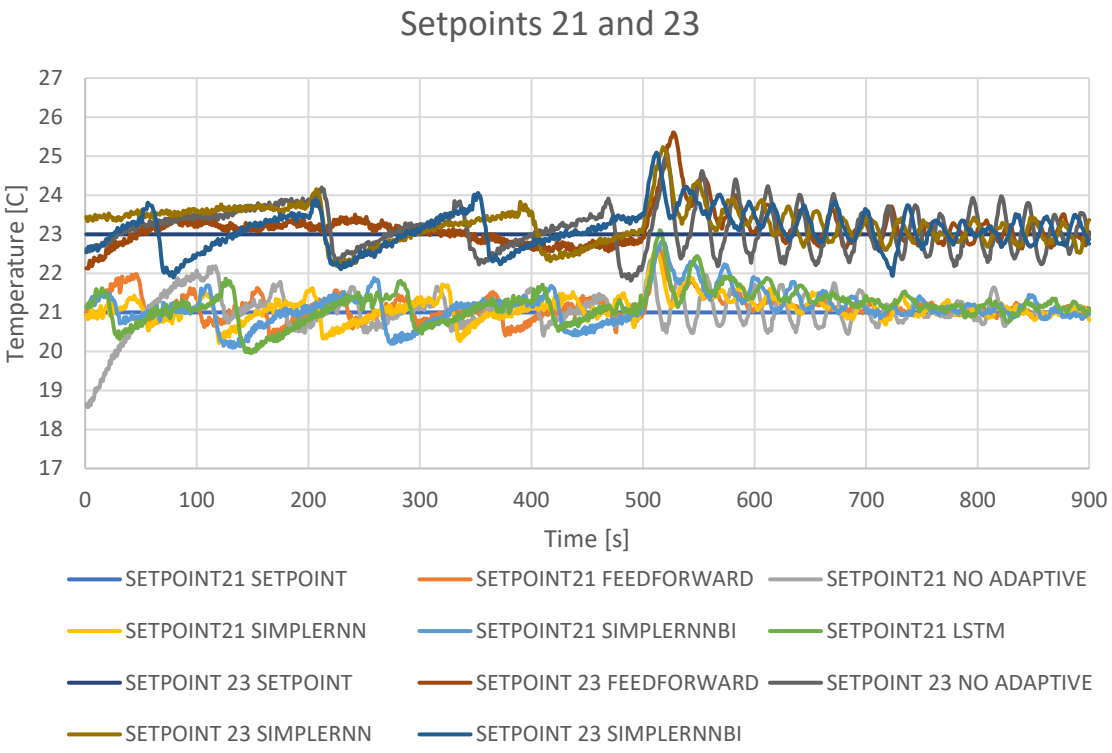


Figure 52 - Control ON/OFF results



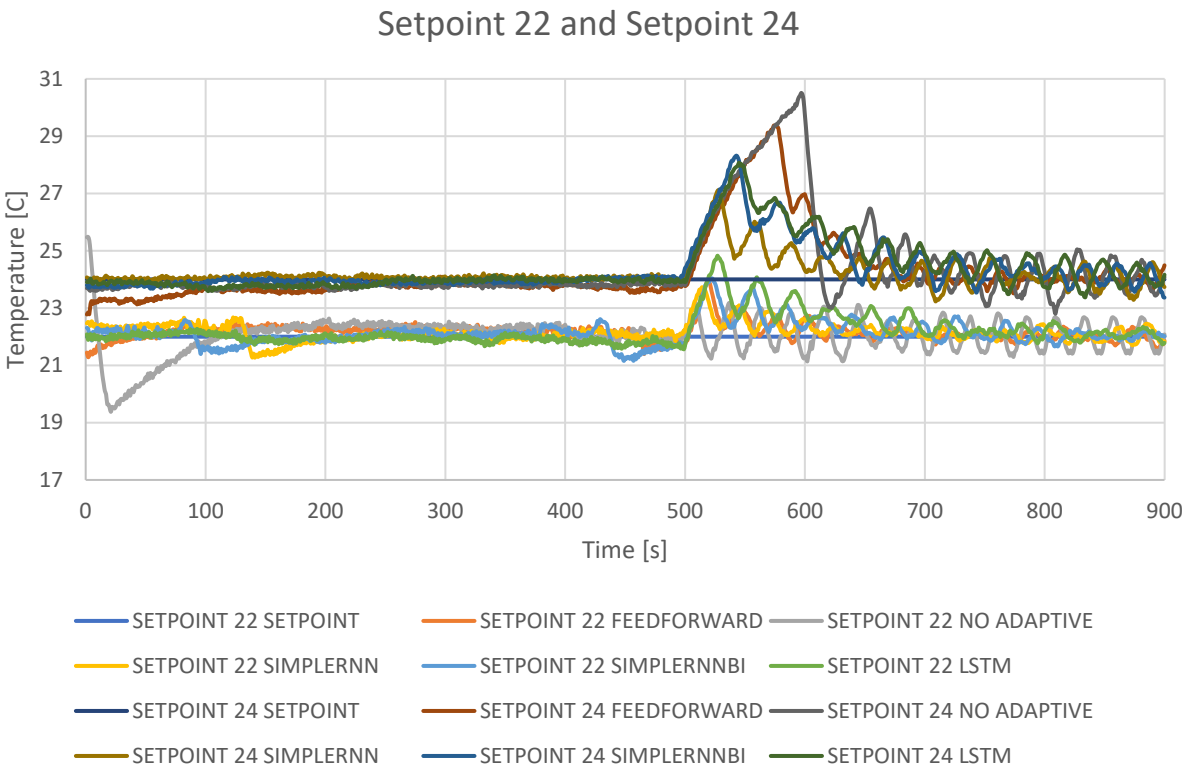
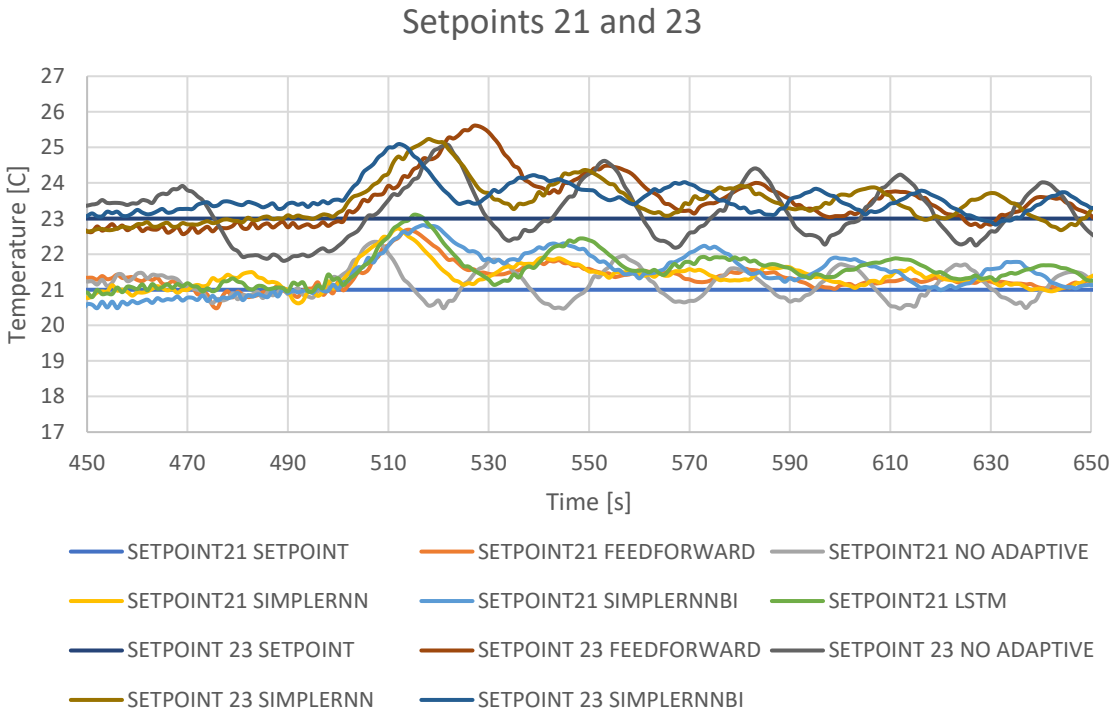


Figure 53 - Experimental Neuro-PI Results



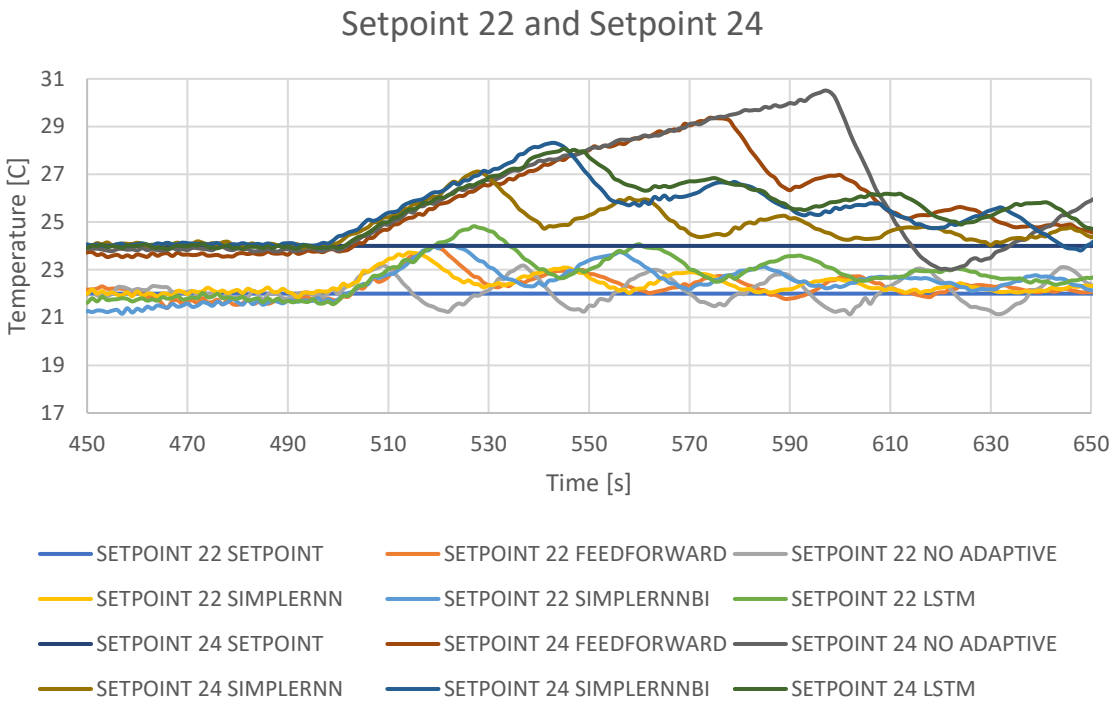


Figure 54 - Results in range 450 - 650 s

From an operational standpoint, this tendency for anticipatory variation provided by recurrent neural networks tends to be beneficial in control using cloud architecture, as it mitigates the delay between cloud processing and the HTTP transmission of control signals to the microcontroller. For example, let's assume a transmission delay of 5 seconds. When adaptive tuning occurs, the control signal will take 5 seconds to reach the microcontroller and act on the system with new proportional and integrative gain values. However, if used recurrent neural networks with a ten-second time step, when the parametric change occurs, the new gain values would have already been transitioning 10 seconds earlier, thus mitigating the 5-second transmission delay.

When comparing the experimental results presented in Figure 53 with the simulation results in Figure 46, it is observed that in the simulation, the temperature varies according to the controller gains adjusted by the machine learning algorithm used. However, in the real experiment, this variation takes longer to occur. This is partly because, in the simulation, the variation occurred to reduce the static gain due to the increase in the air mass flow, whereas in the experiment – with the air mass flow being the manipulated variable – the parametric variation occurred with the increase in thermal load. This shows that the system is more sensitive to variations in air mass flow than to the increase in thermal load and that thermal inertia imposes a slower transition than predicted by the simulations.

Regarding the control at the setpoint of 24 Celsius, presented in Figure 54, in the experiments with adaptive control using Feedforward neural networks and in the experiment with non-adaptive gain control, there was a more significant accumulation of the integrative part than in the other experiments, as shown in Figure 55. This caused the control to take longer to respond to the plant change, a problem known as Windup (Ogata 2001). From the standpoint of mitigating windup, the anticipatory transient in relation to the parametric change tends to be beneficial as it can mitigate this problem.

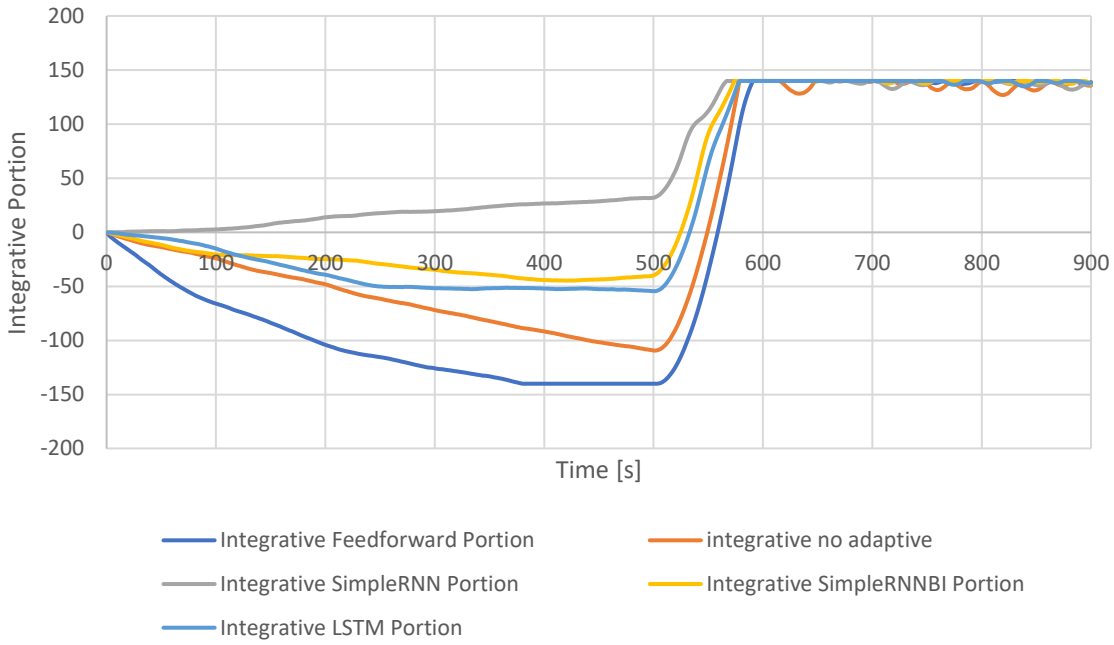


Figure 55 - Integrative Part - Sepoint 24

Regarding the types of algorithms that provided the smallest deviations from the setpoint at the moment of the plant change, Figure 56 presents the standard deviation between seconds 2990 and 3150 for the simulation and between seconds 490 and 650 for the experimental results. These time intervals were chosen because the plant change occurred at second 3000 in the simulation and at second 500 in the experiment. Neither in Figure 56 nor in the results presented for the Bidirectional LSTM. This is because, experimentally, the Bidirectional LSTM became computationally expensive, consuming a lot of memory during training and execution, making it impractical for the experiment. From a simulation standpoint, the results of the LSTM and Bidirectional LSTM showed few differences, indicating that the unidirectional LSTM is sufficiently capable of understanding changes in controller gains in response to plant changes.

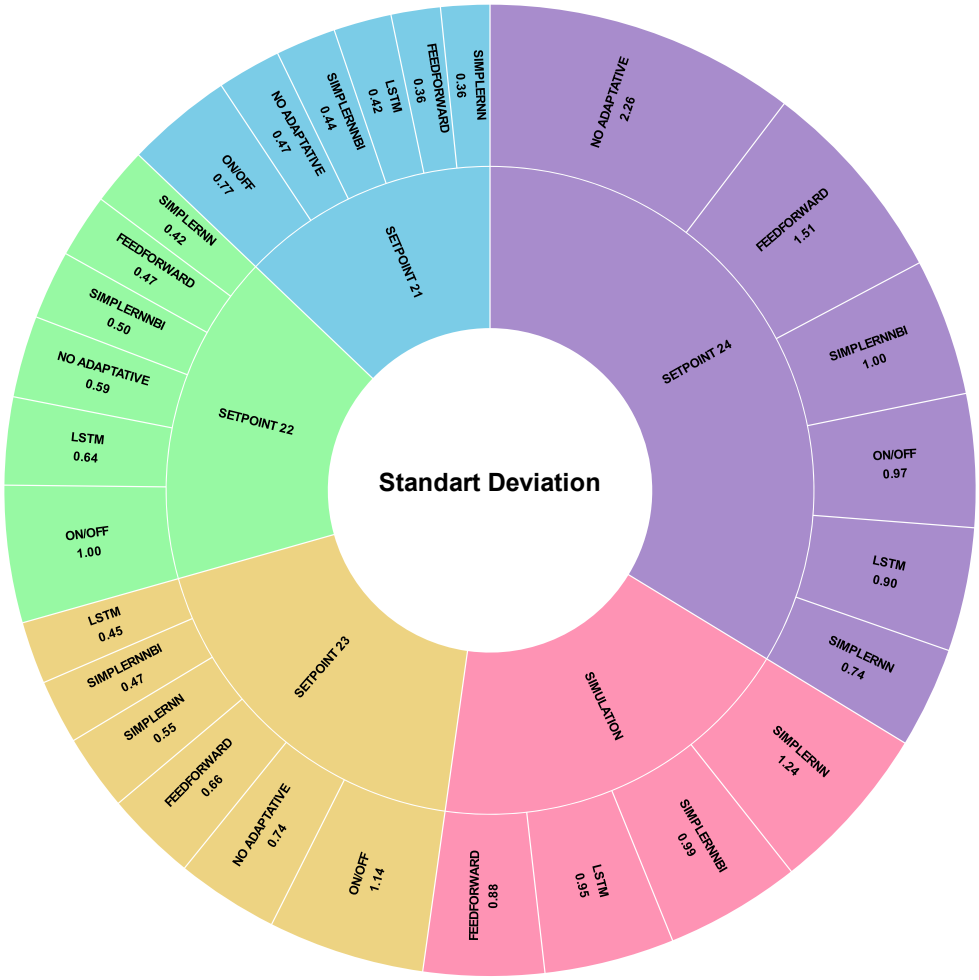


Figure 56 - Standard Deviation for the experimental and simulation results between the seconds for the plants change

Except for the 22°C setpoint experiment, non-adaptive tuned control showed inferior performance, with high standard deviation during the plant change region. Considering the graphs presented in Figure 53 and Figure 54, it is observable that the absence of adaptive tuning caused temperature to oscillate periodically, indicating loss of controllability compared to adaptive methods. This experimental result confirms the simulation results observed in Figure 48. However, unlike Figure 48 where simulation shows increased loss of controllability causing temperature to rise over time, in the experiment, temperature encounters resistance imposed by the HVAC system and oscillates as a consequence.

Summing the standard deviations by model for the four setpoints of the experimental data, ON/OFF control showed the worst results, followed by non-adaptive PI control. Overall, the SimpleRNN without bidirectionality showed the best results. Figure 57 presents the sum of standard deviations per model.

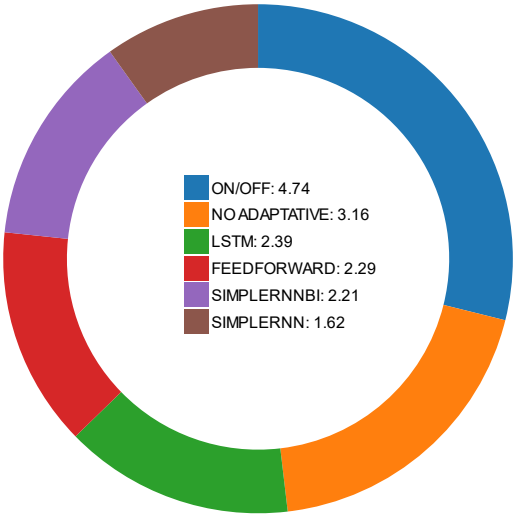


Figure 57 Sum of standard deviation by model

As an addition, and for comparison purposes, the same recurrent deep learning algorithms were tested to directly control the temperature by adjusting the PWM signal based on the error measured relative to the setpoint. The PWM duty cycle ranges from 0 to 100%. Minimum and maximum errors from experimental tests of the adaptive control were measured, and the range between the minimum and maximum errors was used to associate with the duty cycle values from 0 to 100%. The minimum error was associated with 0% and the maximum error with 100%. The neural network used was a deep recurrent neural network with the same structure in hidden layers and hyperparameters as those used in simulation and experimentation, however, with a just one input associated with the error and one output associated with the PWM. It was expected that the network's predictive capability would enable it to foresee the most appropriate PWM value for control. However, the results were inferior to those of the adaptive control. Figure 58 presents the control results for setpoints of 21, 22, 23, and 24 ºC using recurrent neural networks to directly control the temperature in the system.

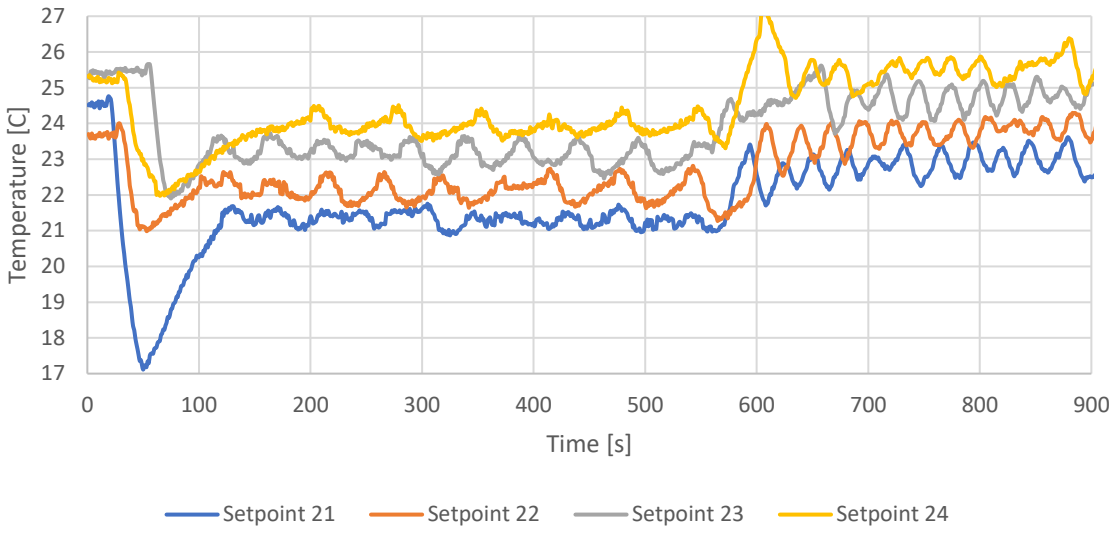


Figure 58 - Directly control Deep Learning

Regarding the direct control results by recurrent neural networks presented in Figure 58, they showed worse results than the adaptive control by deep neural networks. The standard deviations calculated between seconds 490 and 650 were 0.73, 0.77, 0.80, and 1.07 for the setpoints of 21, 22, 23, and 24 °C. These results are worse than those for PI control without neural network adaptation or even equivalent to ON/OFF control. However, direct control by the neural network in adjusting the PWM yielded the worst results in terms of steady-state error. When the thermal load was 40 W, the neural network associated an error value with a PWM value, but when the thermal load changed to 100 W, it continued to associate the error value with the same PWM value. This demonstrates the network's inability to perceive changes in the plants, a perception that is achieved in Neuro-PI control through the tuning of the controller gains. Such observations have also been made for simple neural networks in the works of Zaheer-Uddin and Tudoroiu (2004) e J. Wang and An (2006) .

5 CONCLUSIONS

Initially, in the literature review chapter, a framework was presented through Figure 22 to assist in determining which artificial intelligence technique should be used for controlling Heating, Ventilation, and Air Conditioning (HVAC) systems. Analyzing the framework, both adaptive tuning with artificial neural networks or direct control using deep learning algorithms could be applied for the simulation and the experiment. In fact, the control was simulated using adaptive tuning through deep learning, while experimentally, control was performed using adaptive tuning. For comparison purposes, it was also carried out using deep learning to directly modify the PWM signal in experimental work.

In This study it was compared five Deep Learning models for adaptive tuning of a PI controller applied in temperature control through an HVAC system, using both simulation and real-world experimentation, In addition, for comparison purposes, the experiment also included ON/OFF control, PI control without adaptation, and neural networks directly acting on the modification of the PWM signal. It was observed that direct ON/OFF control on the fan resulted in the poorest control outcomes, as evidenced by higher standard deviations across the four evaluated setpoints. Contrasting experimental results with simulations revealed that while in simulation the temperature varied according to controller gains adjusted by machine learning algorithms, this variability was not significantly observed in the real experiment due to the thermal inertia of the system and greater influence of the air mass flow in the simulations than of the thermal load in the real experiment.

In both, simulation and experiments, controller gains began to be adjusted 10 seconds in advance by recurrent algorithms. Comparing standard deviations between simulation and experiment during specific intervals revealed that the use of Bidirectional LSTM was impractical experimentally due to high computational costs, whereas in simulation, performance between unidirectional and bidirectional LSTM was comparable. This suggests that for many applications, unidirectional LSTM may be sufficient to understand changes in controller gains. Furthermore, bidirectionality did not show significant advantages over unidirectionality in temperature control. However, bidirectionality did increase the speed of model training by reducing the number of training epochs, which can be beneficial in applications requiring real-time model updates.

Results also indicate that methods without adaptive tuning resulted in greater temperature oscillation, highlighting loss of controllability compared to adaptive methods. Adaptive control with recurrent neural networks proved particularly effective in situations

where the plant underwent significant changes, mitigating undesired accumulations in the integral part of the PI control. For example, considering the accumulated results shown in Figure 57, while the non-adaptive control presented a total standard deviation of 3.16 in the experiments, the tuning performed by a recurrent network without LSTM and without bidirectionality showed an accumulated standard deviation of 1.62, representing a 48.73% reduction in the standard deviation.

The use of deep feedforward neural networks showed promising results at specific setpoints, while recurrent neural networks demonstrated advantages in control situations with abrupt plant changes due to their sequential learning capability. Between deep recurrent networks and feedforward networks in real-time applications, considering the shorter training times of recurrent networks and their greater ability to mitigate problems such as windup in the PI controller, as well as their predictive capability being beneficial for mitigating issues with time delay in the wireless transmission signal if used, it becomes preferable to use deep recurrent neural networks over deep feedforward networks. Additionally, considering the standard deviation shown in Figure 57, the recurrent networks without bidirectionality and without LSTM presented a 30% reduction compared to the feedforward algorithm. This indicates that recurrent adaptive tuning is superior to feedforward adaptive tuning.

Regarding the control results by directly adjusting the PWM presented in Figure 58, direct control proved to be as poor as ON/OFF control since the neural network associates an error with a PWM value without considering the plant change. This characteristic has already been observed in the works of Zaheer-Uddin and Tudoroiu (2004) and J. Wang and An (2006) for simple machine learning algorithms, and it has now been described for deep learning algorithms.

6 SUGGESTIONS FOR FUTURE WORK

It has been observed as a control trend to use reinforcement learning algorithms in HVAC system control. However, there are no experimental works on this. Therefore, it is suggested to use the control project built in this thesis to serve as an environment for the training of reinforcement learning models.

The simulation involving the alteration of parameters in the first-order transfer function for temperature variations in environments conditioned by HVAC systems is challenging, as there is limited experimental data available in the technical literature. Therefore, to achieve more accuracy in understanding how temperature varies with changes in the respective systems, considering parameters such as changes in air mass flow rates or thermal loads, it is advisable to use computational fluid dynamics (CFD) simulations.

Cloud control was implemented to leverage the full precision of deep learning algorithms; however, it is also possible to use the same algorithms on the Raspberry Pi 4 through model quantization techniques. This technique is already implemented in commercial tools such as TensorFlow Lite. It would be interesting to compare cloud control with direct control using the Raspberry Pi with model quantization to assess the differences between the two approaches.

The communication protocol used for the Raspberry Pi 4 to communicate with the cloud was HTTP; however, other communication protocols such as MQTT could also be used. A systematic comparative study of the different communication protocols applied to cloud control would be valuable for the field.

The experimental control used in this thesis operated in near real-time, with the only delay being the communication between the cloud and the actuator. In this configuration, bidirectionality did not provide comparative advantages in the control task, as bidirectional algorithms require future data to make decisions, and such data does not exist in real-time applications. However, in systems with slow dynamics, control using batch data could be applied, and in these cases, bidirectional models could be useful. A study applying bidirectional algorithms for control using batch data would be a valuable addition to the technical literature.

The data acquisition was performed using a data acquisition board from National Instruments, and the measured data was transferred to the Raspberry Pi via wireless transmission using the File Transfer Protocol (FTP). However, it would also be possible to perform the acquisition and communication with the Raspberry Pi using Arduino boards and analog/digital converters. This would eliminate the need for the National Instruments

acquisition board and wireless transmission. A comparative study between the two techniques would be interesting.

7 ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior– Brasil (CAPES) – Finance Code 001. Third parties Figures were reproduced according to copyright rules and the Publishers are fully acknowledged.

BIBLIOGRAPHY

Abbassi, A., and L. Bahar. 2005. "Application of Neural Network for the Modeling and Control of Evaporative Condenser Cooling Load." *Applied Thermal Engineering* 25(17–18): 3176–86.

Adesanya, Misbaudeen Aderemi et al. 2024. "Deep Reinforcement Learning for PID Parameter Tuning in Greenhouse HVAC System Energy Optimization: A TRNSYS-Python Cosimulation Approach." *Expert Systems with Applications* 252(PA): 124126. https://doi.org/10.1016/j.eswa.2024.124126.

Afram, Abdul, and Farrokh Janabi-Sharifi. 2014. "Theory and Applications of HVAC Control Systems - A Review of Model Predictive Control (MPC)." *Building and Environment* 72: 343–55.

Aggarwal, Alankrita, Mamta Mittal, and Gopi Battineni. 2021. "Generative Adversarial Network: An Overview of Theory and Applications." *International Journal of Information Management Data Insights* 1(1).

Aguirre, Luis Antonio. 2007. *Introdução a Identificação de Sistemas*. 3rd ed. Belo Horizonte.

Albus J. S. 1971. "A New Approach to Manipulator Control (CMAC)." *Journal of Dynamic Systems, Measurement, and Control*.

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. "Deep Reinforcement Learning: A Brief Survey." *IEEE Signal Processing Magazine* 34(6): 26–38.

Atkeson, Christopher G., and Stefan Schaal. 1995. "Memory-Based Neural Networks for Robot Learning." *Neurocomputing* 9(3): 243–69.

Attaran, Seyed Mohammad, Rubiyah Yusof, and Hazlina Selamat. 2016. "A Novel Optimization Algorithm Based on Epsilon Constraint-RBF Neural Network for Tuning PID Controller in Decoupled HVAC System." *Applied Thermal Engineering* 99: 613–24. http://dx.doi.org/10.1016/j.applthermaleng.2016.01.025.

Aubert, Antoine. 2017. "Multitudes : Aux Origines d'une Revue Radicale." *Raisons Politiques* 67(3): 31–47.

Baur, Christoph et al. 2021. "Autoencoders for Unsupervised Anomaly Segmentation in Brain MR Images: A Comparative Study." *Medical Image Analysis* 69: 101952. https://doi.org/10.1016/j.media.2020.101952.

Bentéjac, Candice, Anna Csörgő, and Gonzalo Martínez-Muñoz. 2021. 54 Artificial Intelligence Review *A Comparative Analysis of Gradient Boosting Algorithms*. Springer Netherlands. https://doi.org/10.1007/s10462-020-09896-5.

Biemann, Marco, Fabian Scheller, Xiufeng Liu, and Lizhen Huang. 2021. "Experimental Evaluation of Model-Free Reinforcement Learning Algorithms for Continuous HVAC Control." *Applied Energy* 298(May): 117164. https://doi.org/10.1016/j.apenergy.2021.117164.

Block, H. D. 1962. "The Perceptron: A Model for Brain Functioning. I." *Reviews of Modern Physics* 34(1): 123–35.

Bouguettaya, Athman et al. 2015. "Efficient Agglomerative Hierarchical Clustering." *Expert Systems with Applications* 42(5): 2785–97.

Bouzaiene, Ramzi, Sami Hafsi, and Faouzi Bouani. 2021. "Adaptive Neural Network PID Controller for Nonlinear Systems." *2021 IEEE 2nd International Conference on Signal, Control and Communication, SCC 2021*: 264–69.

Brandi, Silvio, Marco Savino Piscitelli, Marco Martellacci, and Alfonso Capozzoli. 2020. "Deep Reinforcement Learning to Optimise Indoor Temperature Control and Heating Energy Consumption in Buildings." *Energy and Buildings* 224: 110225. https://doi.org/10.1016/j.enbuild.2020.110225.

Brusey, James, Diana Hintea, Elena Gaura, and Neil Beloe. 2018. “Reinforcement Learning-Based Thermal Comfort Control for Vehicle Cabins.” *Mechatronics* 50: 413–21. https://doi.org/10.1016/j.mechatronics.2017.04.010.

Butz, Martin V., and Oliver Herbort. 2008. “Context-Dependent Predictions and Cognitive Arm Control with XCSF.” *GECCO’08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation 2008*: 1357–64.

Chaudhuri, Tanaya, Yeng Chai Soh, Hua Li, and Lihua Xie. 2019. “A Feedforward Neural Network Based Indoor-Climate Control Framework for Thermal Comfort and Energy Saving in Buildings.” *Applied Energy* 248(April): 44–53. https://doi.org/10.1016/j.apenergy.2019.04.065.

Chen, Yujiao et al. 2020. “Transfer Learning with Deep Neural Networks for Model Predictive Control of HVAC and Natural Ventilation in Smart Buildings.” *Journal of Cleaner Production* 254: 119866. https://doi.org/10.1016/j.jclepro.2019.119866.

Chen, Yujiao, Leslie K. Norford, Holly W. Samuelson, and Ali Malkawi. 2018. “Optimal Control of HVAC and Window Systems for Natural Ventilation through Reinforcement Learning.” *Energy and Buildings* 169: 195–205. https://doi.org/10.1016/j.enbuild.2018.03.051.

Cheung, Yiu Ming. 2003. “K*-Means: A New Generalized k-Means Clustering Algorithm.” *Pattern Recognition Letters* 24(15): 2883–93.

Chiang, Boon, Intan Zaurah Mat Darus, Hishamuddin Jamaluddin, and Haslinda Mohamed Kamar. 2014. “Dynamic Modelling of an Automotive Variable Speed Air Conditioning System Using Nonlinear Autoregressive Exogenous Neural Networks.” *Applied Thermal Engineering* 73(1): 1255–69. http://dx.doi.org/10.1016/j.applthermaleng.2014.08.043.

Choi, Wansik, Jae Woong Kim, Changsun Ahn, and Juhui Gim. 2022. “Reinforcement Learning-Based Controller for Thermal Management System of Electric Vehicles.” *2022 IEEE Vehicle Power and Propulsion Conference, VPPC 2022 - Proceedings*.

Chollet, F., & others. 2015. “Keras.” https://github.com/fchollet/keras.

Clifton, Jesse, and Eric Laber. 2020. “Q-Learning: Theory and Applications.” *Annual Review of Statistics and Its Application* 7: 279–301.

COON, G. H. COHEN & G. A. 1953. “Theoretical Consideration of Retarded Control.” *file:///C:/Users/ramon/OneDrive/Área de Trabalho/artigos/PII_ S0019-0578(07)90000-2-1.pdf*.

Dai, Mingkun, Hangxin Li, and Shengwei Wang. 2023. “A Reinforcement Learning-Enabled Iterative Learning Control Strategy of Air-Conditioning Systems for Building Energy Saving by Shortening the Morning Start Period.” *Applied Energy* 334(July 2022): 120650. https://doi.org/10.1016/j.apenergy.2023.120650.

Dai, Xilei, Siyu Cheng, and Adrian Chong. 2023. “Deciphering Optimal Mixed-Mode Ventilation in the Tropics Using Reinforcement Learning with Explainable Artificial Intelligence.” *Energy and Buildings* 278: 112629. https://doi.org/10.1016/j.enbuild.2022.112629.

Databricks. 2018. “An Open Platform for the Machine Learning Lifecycle.”

Docker. 2024. “Docker.” https://www.docker.com/.

Du, Yan et al. 2021. “Intelligent Multi-Zone Residential HVAC Control Strategy Based on Deep Reinforcement Learning.” *Applied Energy* 281(October 2020): 116117. https://doi.org/10.1016/j.apenergy.2020.116117.

Fang, Xi et al. 2022. “Deep Reinforcement Learning Optimal Control Strategy for Temperature Setpoint Real-Time Reset in Multi-Zone Building HVAC System.” *Applied Thermal Engineering* 212(April): 118552. https://doi.org/10.1016/j.applthermaleng.2022.118552.

FastAPI. 2024. “FastAPI.” https://fastapi.tiangolo.com/reference/.

Ferreira, Pedro M., Sergio M. Silva, and António E. Ruano. 2012. "Model Based Predictive Control of HVAC Systems for Human Thermal Comfort and Energy Consumption Minimisation." *IFAC Proceedings Volumes (IFAC-PapersOnline)* 45(4): 236–41.

Fong, Simon, Saif Ur Rehman, Kamran Aziz, and Information Science. 2014. "DBSCAN : Past , Present and Future." : 232–38.

Frank, Eibe, Leonard Trigg, Geoffrey Holmes, and Ian H. Witten. 2000. "Technical Note: Naive Bayes for Regression." *Machine Learning* 41(1): 5–25.

Ganowicz, Agnieszka, Bartosz Starosta, Aleksandra Knapinska, and Krzysztof Walkowiak. 2022. "Short-Term Network Traffic Prediction with Multilayer Perceptron." *6th SLAAI - International Conference on Artificial Intelligence, SLAAI-ICAI-2022*: 1–6.

Gao, Guanyu, Jie Li, and Yonggang Wen. 2020. "DeepComfort: Energy-Efficient Thermal Comfort Control in Buildings Via Reinforcement Learning." *IEEE Internet of Things Journal* 7(9): 8472–84.

Garnier, Antoine, Julien Eynard, Matthieu Caussanel, and Stéphane Grieu. 2015. "Predictive Control of Multizone Heating, Ventilation and Air-Conditioning Systems in Non-Residential Buildings." *Applied Soft Computing Journal* 37: 847–62. http://dx.doi.org/10.1016/j.asoc.2015.09.022.

Guo, Chenyi, and Qing Song. 2009. "Real-Time Control of Variable Air Volume System Based on a Robust Neural Network Assisted PI Controller." *IEEE Transactions on Control Systems Technology* 17(3): 600–607.

Guo, Youming et al. 2022. "Adaptive Optics Based on Machine Learning: A Review." *Opto-Electronic Advances* 5(7): 1–20.

Haarnoja, Tuomas et al. 2018. "Soft Actor-Critic Algorithms and Applications." http://arxiv.org/abs/1812.05905.

Halhoul Merabet, Ghezlane et al. 2021. "Intelligent Building Control Systems for Thermal Comfort and Energy-Efficiency: A Systematic Review of Artificial Intelligence-Assisted Techniques." *Renewable and Sustainable Energy Reviews* 144(April): 110969. https://doi.org/10.1016/j.rser.2021.110969.

Haskara, Ibrahim, Bharatkumar Hegde, and Chen Fang Chang. 2022. "Reinforcement Learning Based EV Energy Management for Integrated Traction and Cabin Thermal Management Considering Battery Aging." *IFAC-PapersOnLine* 55(24): 348–53. https://doi.org/10.1016/j.ifacol.2022.10.308.

HECHT-NIELSEN, ROBERT. 1992. Neural Networks for Perception *Theory of the Backpropagation Neural Network**Based on "Nonindent" by Robert Hecht-Nielsen, Which Appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE.* Academic Press, Inc. http://dx.doi.org/10.1016/B978-0-12-741252-8.50010-8.

Henze, Gregor P., and Richard E. Hindman. 2002. "Control of Air-Cooled Chiller Condenser Fans Using Clustering Neural Networks." *ASHRAE Transactions* 108 PART 2(1): 232–44.

Hintea, Diana et al. 2014. "Applicability of Thermal Comfort Models to Car Cabin Environments." *ICINCO 2014 - Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* 1: 769–76.

Ho, W. K., O. P. Gan, E. B. Tay, and E. L. Ang. 1996. "Performance and Gain and Phase Margins of Well-Known PID Tuning Formulas." *IEEE Transactions on Control Systems Technology* 4(4): 473–77.

Inc, The MathWorks. 2022. "MATLAB R2022a."

Javed, Abbas, Hadi Larijani, Ali Ahmadinia, and Des Gibson. 2017. "Smart Random Neural Network Controller for HVAC Using Cloud Computing Technology." *IEEE Transactions on Industrial Informatics* 13(1): 351–60.

Jia, Hongjie, Shifei Ding, Xinzheng Xu, and Ru Nie. 2014. “The Latest Research Progress on Spectral Clustering.” *Neural Computing and Applications* 24(7–8): 1477–86.
Jiangjiang Wang, Youyin Jing, and Dawei An. 2006. “Study of Neuron Adaptive PID Controller in a Single-Zone HVAC System.” : 142–45.
Joo, Sungho et al. 2023. “Multi-Agent Reinforcement Learning Based Actuator Control for EV HVAC Systems.” *IEEE Access* 11(October 2022): 7574–87.
Kajino, Yuichi et al. 2000. “Development of Automatic Climate Control with Neural Control.” *SAE Technical Papers* (724).
Ke-Lin Du, Chi-Sing, Wai Ho, Swamy. 2022. “Perceptron: Learning, Generalization, Model Selection, Fault Tolerance, and Role in the Deep Learning Era.” *Mathematics* 10(24): 1–46.
Ketkar, Nikhil. 2017. “Deep Learning with Python.” *Deep Learning with Python*: 113–32.
Khayyam, Hamid, Abbas Z. Kouzani, Eric J. Hu, and Saeid Nahavandi. 2011. “Coordinated Energy Management of Vehicle Air Conditioning System.” *Applied Thermal Engineering* 31(5): 750–64. http://dx.doi.org/10.1016/j.applthermaleng.2010.10.022.
Kotsiantis, S. B. 2013. “Decision Trees: A Recent Overview.” *Artificial Intelligence Review* 39(4): 261–83.
Kranz, J., T. I. Van Niekerk, H. F.G. Holdack-Janssen, and G. Gruhler. 2012. “Automotive Thermal Comfort Control - A Blackbox Approach.” *SAIEE Africa Research Journal* 103(2): 66–76.
Ku, K. L., J. S. Liaw, M. Y. Tsai, and T. S. Liu. 2015. “Automatic Control System for Thermal Comfort Based on Predicted Mean Vote and Energy Saving.” *IEEE Transactions on Automation Science and Engineering* 12(1): 378–83.
LaValley, Michael P. 2008. “Logistic Regression.” *Circulation* 117(18): 2395–99.
Lee, Dasheng, and Shang Tse Lee. 2023. “Artificial Intelligence Enabled Energy-Efficient Heating, Ventilation and Air Conditioning System: Design, Analysis and Necessary Hardware Upgrades.” *Applied Thermal Engineering* 235(May): 121253. https://doi.org/10.1016/j.applthermaleng.2023.121253.
Lee, Seung-chul, and Wan-young Chung. 2005. “Intelligent Air Quality Sensor System with Back Propagation Neural Network in Automobile 2 . INTELLIGENT AQS SYSTEM 3 . GAS SENSING PROPERTIES OF THE USED 4 . NEURON NETWORK FOR INTELLIGENT.” : 468–71.
Li, Ning et al. 2012. “Dynamic Modeling and Control of a Direct Expansion Air Conditioning System Using Artificial Neural Network.” *Applied Energy* 91(1): 290–300. http://dx.doi.org/10.1016/j.apenergy.2011.09.037.
———. 2013. “On-Line Adaptive Control of a Direct Expansion Air Conditioning System Using Artificial Neural Network.” *Applied Thermal Engineering* 53(1): 96–107. http://dx.doi.org/10.1016/j.applthermaleng.2013.01.008.
Li, Xiuming, Tianyi Zhao, Jili Zhang, and Tingting Chen. 2017. “Predication Control for Indoor Temperature Time-Delay Using Elman Neural Network in Variable Air Volume System.” *Energy and Buildings* 154: 545–52. https://doi.org/10.1016/j.enbuild.2017.09.005.
Lillicrap, Timothy P., and Adam Santoro. 2019. “Backpropagation through Time and the Brain.” *Current Opinion in Neurobiology* 55: 82–89. https://doi.org/10.1016/j.conb.2019.01.011.
Liu, Boming, Murat Akcakaya, and Thomas E. McDermott. 2021. “Automated Control of Transactive HVACs in Energy Distribution Systems.” *IEEE Transactions on Smart Grid* 12(3): 2462–71.
Liu, Chenyu et al. 2022. “Partial Least Squares Regression and Principal Component Analysis: Similarity and Differences between Two Popular Variable Reduction

Approaches.” *General Psychiatry* 35(1): 1–5.
Liu, Shujie, Nan Yang, Mu Li, and Ming Zhou. 2014. “A Recursive Recurrent Neural Network for Statistical Machine Translation.” *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference* 1: 1491–1500.
Macarulla, Marcel, Miquel Casals, Núria Forcada, and Marta Gangolells. 2017. “Implementation of Predictive Control in a Commercial Building Energy Management System Using Neural Networks.” *Energy and Buildings* 151: 511–19. http://dx.doi.org/10.1016/j.enbuild.2017.06.027.
Mao, Shitong, and Ervin Sejdic. 2022. “A Review of Recurrent Neural Network-Based Methods in Computational Physiology.” *IEEE Transactions on Neural Networks and Learning Systems*: 1–21.
Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo et al. 2015. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,.”
Matsuo, Yutaka et al. 2022. “Deep Learning, Reinforcement Learning, and World Models.” *Neural Networks* 152: 267–75. https://doi.org/10.1016/j.neunet.2022.03.037.
Maurício, José, Inês Domingues, and Jorge Bernardino. 2023. “Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review.” *Applied Sciences (Switzerland)* 13(9).
Mendes, Ramon de Paoli, Juan José Garcia Pábon, Daniel Leon Ferreira Pottie, and Luiz Machado. 2024. “Artificial Intelligence Strategies Applied in General and Automotive Air Conditioning Control. A Review of the Last 20 Years.” *International Journal of Refrigeration* 164(March): 180–98.
Montreal, U. 2013. “ADVANCES IN OPTIMIZING RECURRENT NETWORKS Yoshua Bengio , Nicolas Boulanger-Lewandowski and Razvan Pascanu.” *Integration The Vlsi Journal*: 8624–28.
Moon, Jin Woo, and Sung Kwon Jung. 2016. “Development of a Thermal Control Algorithm Using Artificial Neural Network Models for Improved Thermal Comfort and Energy Efficiency in Accommodation Buildings.” *Applied Thermal Engineering* 103: 1135–44. http://dx.doi.org/10.1016/j.applthermaleng.2016.05.002.
Nagarathinam, Srinarayana, Vishnu Menon, Arunchandar Vasan, and Anand Sivasubramaniam. 2020. “MARCO - Multi-Agent Reinforcement Learning Based COntrol of Building HVAC Systems.” *e-Energy 2020 - Proceedings of the 11th ACM International Conference on Future Energy Systems* (Ml): 57–67.
Nanayakkara, Visakha K., Yasuyuki Ikegami, and Haruo Uehara. 2002. “Evolutionary Design of Dynamic Neural Networks for Evaporated Control.” *International Journal of Refrigeration* 25(6): 813–26.
Nayak, Dillip Ranjan et al. 2022. “Brain Tumor Classification Using Dense Efficient-Net.” *Axioms* 11(1).
Neves, A. C., Ignacio González, John Leander, and Raid Karoumi. 2018. “A New Approach to Damage Detection in Bridges Using Machine Learning.” *Lecture Notes in Civil Engineering* 5(October): 73–84.
Ng, Boon Chiang, Intan Zaurah Mat Darus, Hishamuddin Jamaluddin, and Haslinda Mohamed Kamar. 2014. “Application of Adaptive Neural Predictive Control for an Automotive Air Conditioning System.” *Applied Thermal Engineering* 73(1): 1244–54. http://dx.doi.org/10.1016/j.applthermaleng.2014.08.044.
Ogata, K. 2001. *Modern Control Engineering*. 4 Edição. ed. Prentice-Hall.
Org, Tensorflow. “Tensorflow Lite.” https://www.tensorflow.org/lite?hl=pt-br.
de Paoli Mendes, Ramon. 2020. “Belo Horizonte 2020.”
Parmar, Aakash, Rakesh Katariya, and Vatsal Patel. 2019. “A Review on Random Forest: An

Ensemble Classifier." *Lecture Notes on Data Engineering and Communications Technologies* 26: 758–63.
Peng, Kai, Victor C.M. Leung, and Qingjia Huang. 2018. "Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection System over Big Data." *IEEE Access* 6(c): 11897–906.
Pham, Binh Thai, Dieu Tien Bui, and Indra Prakash. 2018. "Bagging Based Support Vector Machines for Spatial Prediction of Landslides." *Environmental Earth Sciences* 77(4). https://doi.org/10.1007/s12665-018-7268-y.
Png, Ethan et al. 2019. "An Internet of Things Upgrade for Smart and Scalable Heating, Ventilation and Air-Conditioning Control in Commercial Buildings." *Applied Energy* 239(September 2018): 408–24. https://doi.org/10.1016/j.apenergy.2019.01.229.
Postgresql. 2024. "PostgreSQL."
Qin, Haosen et al. 2023. "Energy-Efficient Heating Control for Nearly Zero Energy Residential Buildings with Deep Reinforcement Learning." *Energy* 264(November 2022).
Qiu, Shunian et al. 2020. "Model-Free Control Method Based on Reinforcement Learning for Building Cooling Water Systems: Validation by Measured Data-Based Simulation." *Energy and Buildings* 218.
Ramchoun, Hassan et al. 2016. "Multilayer Perceptron: Architecture Optimization and Training." *International Journal of Interactive Multimedia and Artificial Intelligence* 4(1): 26.
Razi, M., M. Farrokhi, M. H. Saeidi, and A. R.Faghih Khorasani. 2006. "Neuro-Predictive Control for Automotive Air Conditioning System." *IEEE International Conference on Engineering of Intelligent Systems, ICEIS 2006*.
Reichensdörfer, Elias, Johannes Günther, and Klaus Diepold. 2017. "Recurrent Neural Networks for PID Auto-Tuning." : 84.
Research Rabbit. 2021. "Research Rabbit." https://www.researchrabbit.ai/.
Rivera, Daniel E., Manfred Morarl, and Sigurd Skogestad. 1986. "Internal Model Control: Pid Controller Design." *Industrial and Engineering Chemistry Process Design and Development* 25(1): 252–65.
Ruder, Sebastian. 2016. "An Overview of Gradient Descent Optimization Algorithms." : 1–14. http://arxiv.org/abs/1609.04747.
Salavati, Saeed, Karolos Grigoriadis, and Matthew Franchek. 2022. "An Explicit Robust Stability Condition for Uncertain Time-Varying First-Order plus Dead-Time Systems." *ISA Transactions* 126: 171–79. https://doi.org/10.1016/j.isatra.2021.07.046.
Schuh, Günther et al. 2020. "Industrie 4.0 Maturity Index: Managing the Digital Transformation of Companies - UPDATE 2020." *Acatech Study*. https://www.acatech.de/publikation/industrie-4-0-maturity-index-update-2020/download-pdf?lang=en.
Schulman, John et al. 2017. "Proximal Policy Optimization Algorithms." : 1–12. http://arxiv.org/abs/1707.06347.
Shin, Minjae et al. 2024. "Development of an HVAC System Control Method Using Weather Forecasting Data with Deep Reinforcement Learning Algorithms." *Building and Environment* 248(June 2023): 111069. https://doi.org/10.1016/j.buildenv.2023.111069.
Shmilovici, Armin. 2010. "Data Mining and Knowledge Discovery Handbook." *Data Mining and Knowledge Discovery Handbook*.
Silver, David et al. 2014. "Deterministic Policy Gradient Algorithms." *31st International Conference on Machine Learning, ICML 2014* 1: 605–19.
Sinha, Debjyoti, and Mohamed El-Sharkawy. 2019. "Thin MobileNet: An Enhanced MobileNet Architecture." *2019 IEEE 10th Annual Ubiquitous Computing, Electronics*

and Mobile Communication Conference, UEMCON 2019: 0280–85.
Smagulova, Kamilya, and Alex Pappachen James. 2019. "A Survey on LSTM Memristive Neural Network Architectures and Applications." *European Physical Journal: Special Topics* 228(10): 2313–24.
Sohn, Insoo. 2021. "Deep Belief Network Based Intrusion Detection Techniques: A Survey." *Expert Systems with Applications* 167(December 2019): 114170. https://doi.org/10.1016/j.eswa.2020.114170.
Song, Zhihang, Bruce T. Murray, and Bahgat Sammakia. 2014. "A Dynamic Compact Thermal Model for Data Center Analysis and Control Using the Zonal Method and Artificial Neural Networks." *Applied Thermal Engineering* 62(1): 48–57. http://dx.doi.org/10.1016/j.applthermaleng.2013.09.006.
Soudari, Mallikarjun et al. 2016. "Learning Based Personalized Energy Management Systems for Residential Buildings." *Energy and Buildings* 127: 953–68. http://dx.doi.org/10.1016/j.enbuild.2016.05.059.
Van Steenkiste, Sjoerd, Klaus Greff, Michael Chang, and Jürgen Schmidhuber. 2018. "Relational Neural Expectation Maximization: Unsupervised Discovery of Objects and Their Interactions." *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*: 1–15.
Stoecker, W. F., and P. A. Stoecker. 1989. Microcomputer Control of Thermal and Mechanical Systems *Microcomputer Control of Thermal and Mechanical Systems*.
Sun, Shiliang, and Rongqing Huang. 2010. "An Adaptive K-Nearest Neighbor Algorithm." *Proceedings - 2010 7th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010* 1(Fskd): 91–94.
Sun, Yuying et al. 2020. "Development of an Optimal Control Method of Chilled Water Temperature for Constant-Speed Air-Cooled Water Chiller Air Conditioning Systems." *Applied Thermal Engineering* 180(July): 115802. https://doi.org/10.1016/j.applthermaleng.2020.115802.
Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. "Sequence to Sequence Learning with Neural Networks." *Advances in Neural Information Processing Systems* 4(January): 3104–12.
Świechowski, Maciej, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2023. 56 Artificial Intelligence Review *Monte Carlo Tree Search: A Review of Recent Modifications and Applications*. Springer Netherlands. https://doi.org/10.1007/s10462-022-10228-y.
Tallec, Corentin, and Yann Ollivier. 2017. "Unbiasing Truncated Backpropagation Through Time." : 1–13. http://arxiv.org/abs/1705.08209.
Tan, Nursret. 2004. "Computation of Stabilizing PI and PID Controllers for Processes with Time Delay."
Taniguchi, Ueda and Y. 1999. "The Prediction of the Passenger's Thermal Sensation Level Using a Neural Network and Its Application to the Automobile HVAC Control." *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics* 4: 623–34.
Teeter, Jason, and Mo Yuen Chow. 1998. "Application of Functional Link Neural Network to Hvac Thermal Dynamic System Identification." *IEEE Transactions on Industrial Electronics* 45(1): 170–76.
Torres, José F. et al. 2021. "Deep Learning for Time Series Forecasting: A Survey." *Big Data* 9(1): 3–21.
Torres, Jose Luis, and Marcelo Luis Martin. 2008. "Adaptive Control of Thermal Comfort Using Neural Networks." *Argentine Symposium on Computing Technology*. https://www.researchgate.net/publication/268274253.

Torres, Luis, and Marcelo Luis Martin. 2008. “Neural Control of Thermal Comfort Considering User Vote.”

UNEP. 2019. *IEA. 2019 Global Status Report for Buildings and Construction.*

Varshney, Tarun, and Satya Sheel. 2011. “A New Online Tuning Approach for Pid Control of Multivariable Systems Using Diagonal Recurrent Neural Network.” *Proceedings - 2011 IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2011*: 317–20.

de Ville, Barry. 2013. “Decision Trees.” *Wiley Interdisciplinary Reviews: Computational Statistics* 5(6): 448–55.

Visioli, A. 2003. “Modified Anti-Windup Scheme for PID Controllers.” *IEE Proceedings - Control Theory and Applications* 150(1).

Voulodimos, Athanasios, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. 2018. “Deep Learning for Computer Vision: A Brief Review.” *Computational Intelligence and Neuroscience* 2018.

Wang, Fei et al. 2019. “Feature Learning Viewpoint of Adaboost and a New Algorithm.” *IEEE Access* 7: 149890–99.

Wang, Jiangjiang, and Dawei An. 2006. “Simulation and Experiment Study of Neural Network PID Controller in Central Air-Conditioning System.” *2006 IEEE Conference on Cybernetics and Intelligent Systems*.

Wang, Jiangjiang, Chunfa Zhang, and Youyin Jing. 2007. “Hybrid CMAC-PID Controller in Heating Ventilating and Air-Conditioning System.” *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation, ICMA 2007*: 3706–11.

Wei, Tianshu, Yanzhi Wang, and Qi Zhu. 2017. “Deep Reinforcement Learning for Building HVAC Control.” *Proceedings - Design Automation Conference* Part 12828(2).

Wythoff, Barry J. 1993. “Backpropagation Neural Networks. A Tutorial.” *Chemometrics and Intelligent Laboratory Systems* 18(2): 115–55.

Xiao, Aoran et al. 2023. “Unsupervised Point Cloud Representation Learning with Deep Neural Networks: A Survey.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45(9): 11321–39.

Xu, Na et al. 2023. “Application of Self-Organizing Maps to Coal Elemental Data.” *International Journal of Coal Geology* 277(September).

Yamazaki, Takanori, Yuji Yamakawa, Kazuyuki Kamimura, and Shigeru Kurosu. 2011. “Air-Conditioning PID Control System with Adjustable Reset to Offset Thermal Loads Upsets.” *Advances in PID Control*.

Yang, Qinmin, Jianhua Zhu, Xiangguo Xu, and Jiangang Lu. 2016. “Simultaneous Control of Indoor Air Temperature and Humidity for a Chilled Water Based Air Conditioning System Using Neural Networks.” *Energy and Buildings* 110: 159–69. http://dx.doi.org/10.1016/j.enbuild.2015.10.034.

Yao, Ye, Zhiwei Lian, Zhijian Hou, and Weiwei Liu. 2006. “An Innovative Air-Conditioning Load Forecasting Model Based on RBF Neural Network and Combined Residual Error Correction.” *International Journal of Refrigeration* 29(4): 528–38.

Zafar, Afia et al. 2022. “A Comparison of Pooling Methods for Convolutional Neural Networks.” *Applied Sciences (Switzerland)* 12(17): 1–21.

Zaheer-Uddin, M., and N. Tudoroiu. 2004. “Neuro-PID Tracking Control of a Discharge Air Temperature System.” *Energy Conversion and Management* 45(15–16): 2405–15.

Zhang, Zhiang et al. 2019. “Whole Building Energy Model for HVAC Optimal Control: A Practical Framework Based on Deep Reinforcement Learning.” *Energy and Buildings* 199: 472–90. https://doi.org/10.1016/j.enbuild.2019.07.029.

Zhang, Zhiang, and Khee Poh Lam. 2018. “Practical Implementation and Evaluation of Deep Reinforcement Learning Control for a Radiant Heating System.” *BuildSys 2018 -*

Proceedings of the 5th Conference on Systems for Built Environments: 148–57.

Zhang, Zidong, Dongxia Zhang, and Robert C. Qiu. 2020. "Deep Reinforcement Learning for Power System Applications: An Overview." *CSEE Journal of Power and Energy Systems* 6(1): 213–25.

Zhao, Dongbin, Haitao Wang, Kun Shao, and Yuanheng Zhu. 2017. "Deep Reinforcement Learning with Experience Replay Based on SARSA." *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*.

Zhou, Yan. 2024. "A YOLO-NL Object Detector for Real-Time Detection." *Expert Systems with Applications* 238(PE): 122256. https://doi.org/10.1016/j.eswa.2023.122256.

Ziegler, J. G., and N. B. Nichols. 1993. "Optimum Settings for Automatic Controllers." *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME* 115(2B): 220–22.

Zou, Zhengbo, Xinran Yu, and Semiha Ergan. 2020. "Towards Optimal Control of Air Handling Units Using Deep Reinforcement Learning and Recurrent Neural Network." *Building and Environment* 168(October 2019): 106535. https://doi.org/10.1016/j.buildenv.2019.106535.

APPENDIX

1 Dense Neural Network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import Callback
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt

# Importing input data from Excel
df_input = pd.read_excel('INPUT.xlsx')

# Importing output data from Excel
df_output = pd.read_excel('OUTPUT.xlsx')

# Converting input data to a numpy array
X_train = df_input.values

# Converting output data to a numpy array
y_train = df_output.values

# Converting sequences to numpy arrays
X0 = np.array(X_train)
Y0 = np.array(y_train)

# Normalize the input data with L2
X = X0 / np.linalg.norm(X0, ord=2, axis=0, keepdims=True)
Y = Y0 / np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

# Defining the sequential model
model = Sequential()

# Adding the three Dense layers
model.add(Dense(360, activation="relu"))
model.add(Dense(360, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(2, activation="linear"))

# Stop based on error
class StopTrainingOnMSE(Callback):
    def __init__(self, mse_threshold=0.0000003):
        super(StopTrainingOnMSE, self).__init__()
        self.mse_threshold = mse_threshold

    def on_epoch_end(self, epoch, logs=None):
```

```
        current_mse = logs.get('mse')  # Ensure 'mse' matches the metric used
        if current_mse is not None and current_mse < self.mse_threshold:
            self.model.stop_training = True
            print(f"Training stopped. MSE ({current_mse}) is below the threshold
({self.mse_threshold}).")

# Learning rate decay callback
class DecayLearningRate(tf.keras.callbacks.Callback):
    def __init__(self, factor=0.97, patience=25):
        super(DecayLearningRate, self).__init__()
        self.factor = factor
        self.patience = patience
        self.best_loss = np.inf
        self.wait = 0

    def on_epoch_end(self, epoch, logs=None):
        current_loss = logs.get('loss')
        if current_loss < self.best_loss:
            self.best_loss = current_loss
            self.wait = 0
        else:
            self.wait += 1
            if self.wait >= self.patience:
                lr = tf.keras.backend.get_value(self.model.optimizer.lr)
                tf.keras.backend.set_value(self.model.optimizer.lr, lr * self.factor)
                print("Learning rate reduced to:",
tf.keras.backend.get_value(self.model.optimizer.lr))
                self.wait = 0

# Define the custom stopping criterion callback
stop_on_mse_callback = StopTrainingOnMSE(mse_threshold=0.0000003)

# Apply the callback
callback = DecayLearningRate(factor=0.97, patience=25)

# Compiling the model
model.compile(optimizer=Adam(0.001), loss='mean_squared_error', metrics=['mse'])

# Training the model
history_callback = model.fit(X, Y, epochs=5000, batch_size=250, callbacks=[callback,
stop_on_mse_callback], validation_split=0.2)

# Saving the model to a .h5 file
model.save("modelo_ANN.h5")

loss, mse = model.evaluate(X, Y)
print("Loss:", loss)

# Predicting results
result = model.predict(X)
```

```
result2 = result * np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

# Saving the results to an Excel file
df_result = pd.DataFrame(result2, columns=['Column1', 'Column2'])
df_result.to_excel('result.xlsx', index=False)

# Plotting the MSE metric over time
plt.plot(history_callback.history['mse'], label='MSE (Training)')
plt.plot(history_callback.history['val_mse'], label='MSE (Validation)')
plt.title('MSE across epochs')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

2. Recurrente Neural Network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, LSTM, Bidirectional,
GRU
from tensorflow.keras.optimizers import Adam, SGD, Adamax
from sklearn.preprocessing import Normalizer
import numpy as np
import pandas as pd
import time
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import Callback
import matplotlib.pyplot as plt

# TIME STEP
step = 10

# Importing input data from Excel
df_input = pd.read_excel('INPUT.xlsx')

# Importing output data from Excel
df_output = pd.read_excel('OUTPUT.xlsx')

# Converting input data to a numpy array
X_train = df_input.values

# Converting output data to a numpy array
y_train = df_output.values

# Organizing the data into sequences
seq_length = step  # Sequence length
num_samples = X_train.shape[0] - seq_length + 1
```

```
X_sequences = []
y_sequences = []

for i in range(num_samples):
    X_sequences.append(X_train[i:i+seq_length])  # Input sequences
    y_sequences.append(y_train[i+seq_length-1])  # Output sequences (only the last time step)

# Converting the sequences to numpy arrays
X0 = np.array(X_sequences)
Y0 = np.array(y_sequences)

# Normalize the input data with L2
X = X0 / np.linalg.norm(X0, ord=2, axis=0, keepdims=True)
Y = Y0 / np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

# Defining the sequential model
model = Sequential()

# SimpleRNN Model
model.add(SimpleRNN(128, activation="tanh", return_sequences=True, input_shape=(step,
2)))
model.add(SimpleRNN(128, activation="tanh", return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(2, activation="linear"))

# Stop based on error
class StopTrainingOnMSE(Callback):
    def __init__(self, mse_threshold=0.0000003):
        super(StopTrainingOnMSE, self).__init__()
        self.mse_threshold = mse_threshold

    def on_epoch_end(self, epoch, logs=None):
        current_mse = logs.get('mse')  # Ensure 'mse' matches the metric used
        if current_mse is not None and current_mse < self.mse_threshold:
            self.model.stop_training = True
            print(f"Training stopped. MSE ({current_mse}) is below the threshold
({self.mse_threshold}).")

# Patience callback
class DecayLearningRate(tf.keras.callbacks.Callback):
    def __init__(self, factor=0.97, patience=100):
        super(DecayLearningRate, self).__init__()
        self.factor = factor
        self.patience = patience
        self.best_loss = np.inf
        self.wait = 0

    def on_epoch_end(self, epoch, logs=None):
        current_loss = logs.get('loss')
        if current_loss < self.best_loss:
```

```
            self.best_loss = current_loss
            self.wait = 0
        else:
            self.wait += 1
            if self.wait >= self.patience:
                lr = tf.keras.backend.get_value(self.model.optimizer.lr)
                tf.keras.backend.set_value(self.model.optimizer.lr, lr * self.factor)
                print("Learning rate reduced to:",
tf.keras.backend.get_value(self.model.optimizer.lr))
                self.wait = 0

# Apply the patience callback
callback = DecayLearningRate(factor=0.97, patience=100)

# Define the custom stopping criterion callback
stop_on_mse_callback = StopTrainingOnMSE(mse_threshold=0.0000003)

# Compiling the model
model.compile(optimizer=Adam(0.001), loss='mean_squared_error', metrics=['mse'])

# Training the model
history_callback = model.fit(X, Y, epochs=5000, batch_size=800, callbacks=[callback,
stop_on_mse_callback], validation_split=0.2)

# Saving the model to a .h5 file
model.save(f"model_SimpleRNN{step}.h5")

loss, mse = model.evaluate(X, Y)

print("Loss:", loss)

# Predicting results
result = model.predict(X)
result2 = result * np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

# Saving the results to an Excel file
df_result = pd.DataFrame(result2, columns=['Column1', 'Column2'])
df_result.to_excel('result.xlsx', index=False)

# Plotting the MSE metric over time
plt.plot(history_callback.history['mse'], label='MSE (Training)')
plt.plot(history_callback.history['val_mse'], label='MSE (Validation)')
plt.title('MSE across epochs')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

3. Recurrent LSTM Neural Network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, LSTM, Bidirectional,
GRU
from tensorflow.keras.optimizers import Adam, SGD, Adamax
from sklearn.preprocessing import Normalizer
import numpy as np
import pandas as pd
import time
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import Callback
import matplotlib.pyplot as plt

# TIME STEP
step = 10

# Importing input data from Excel
df_input = pd.read_excel('INPUT.xlsx')

# Importing output data from Excel
df_output = pd.read_excel('OUTPUT.xlsx')

# Converting input data to a numpy array
X_train = df_input.values

# Converting output data to a numpy array
y_train = df_output.values

# Organizing the data into sequences
seq_length = step  # Sequence length
num_samples = X_train.shape[0] - seq_length + 1

X_sequences = []
y_sequences = []

for i in range(num_samples):
    X_sequences.append(X_train[i:i+seq_length])  # Input sequences
    y_sequences.append(y_train[i+seq_length-1])  # Output sequences (only the last time step)

# Converting the sequences to numpy arrays
X0 = np.array(X_sequences)
Y0 = np.array(y_sequences)

# Normalize the input data with L2
X = X0 / np.linalg.norm(X0, ord=2, axis=0, keepdims=True)
Y = Y0 / np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

# Defining the sequential model
model = Sequential()
```

```
# LSTM Model
model.add(LSTM(128, activation="tanh", return_sequences=True, input_shape=(step, 2)))
model.add(LSTM(128, activation="tanh", return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(2, activation="linear"))

# Stop based on error
class StopTrainingOnMSE(Callback):
    def __init__(self, mse_threshold=0.0000003):
        super(StopTrainingOnMSE, self).__init__()
        self.mse_threshold = mse_threshold

    def on_epoch_end(self, epoch, logs=None):
        current_mse = logs.get('mse')  # Ensure 'mse' matches the metric used
        if current_mse is not None and current_mse < self.mse_threshold:
            self.model.stop_training = True
            print(f"Training stopped. MSE ({current_mse}) is below the threshold
({self.mse_threshold}).")

# Patience callback
class DecayLearningRate(tf.keras.callbacks.Callback):
    def __init__(self, factor=0.97, patience=100):
        super(DecayLearningRate, self).__init__()
        self.factor = factor
        self.patience = patience
        self.best_loss = np.inf
        self.wait = 0

    def on_epoch_end(self, epoch, logs=None):
        current_loss = logs.get('loss')
        if current_loss < self.best_loss:
            self.best_loss = current_loss
            self.wait = 0
        else:
            self.wait += 1
            if self.wait >= self.patience:
                lr = tf.keras.backend.get_value(self.model.optimizer.lr)
                tf.keras.backend.set_value(self.model.optimizer.lr, lr * self.factor)
                print("Learning rate reduced to:",
tf.keras.backend.get_value(self.model.optimizer.lr))
                self.wait = 0

# Apply the patience callback
callback = DecayLearningRate(factor=0.97, patience=100)

# Define the custom stopping criterion callback
stop_on_mse_callback = StopTrainingOnMSE(mse_threshold=0.0000003)

# Compiling the model
```

```
model.compile(optimizer=Adam(0.001), loss='mean_squared_error', metrics=['mse'])

# Training the model
history_callback = model.fit(X, Y, epochs=5000, batch_size=800, callbacks=[callback,
stop_on_mse_callback], validation_split=0.2)

# Saving the model to a .h5 file
model.save(f"model_LSTM{step}.h5")

loss, mse = model.evaluate(X, Y)

print("Loss:", loss)

# Predicting results
result = model.predict(X)
result2 = result * np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

# Saving the results to an Excel file
df_result = pd.DataFrame(result2, columns=['Column1', 'Column2'])
df_result.to_excel('result.xlsx', index=False)

# Plotting the MSE metric over time
plt.plot(history_callback.history['mse'], label='MSE (Training)')
plt.plot(history_callback.history['val_mse'], label='MSE (Validation)')
plt.title('MSE across epochs')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

4. Bidirectional LSTM

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, LSTM, Bidirectional,
GRU
from tensorflow.keras.optimizers import Adam, SGD, Adamax
from sklearn.preprocessing import Normalizer
import numpy as np
import pandas as pd
import time
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import Callback
import matplotlib.pyplot as plt

# TIME STEP
step = 10

# Importing input data from Excel
df_input = pd.read_excel('INPUT.xlsx')
```

```
# Importing output data from Excel
df_output = pd.read_excel('OUTPUT.xlsx')

# Converting input data to a numpy array
X_train = df_input.values

# Converting output data to a numpy array
y_train = df_output.values

# Organizing the data into sequences
seq_length = step  # Sequence length
num_samples = X_train.shape[0] - seq_length + 1

X_sequences = []
y_sequences = []

for i in range(num_samples):
    X_sequences.append(X_train[i:i+seq_length])  # Input sequences
    y_sequences.append(y_train[i+seq_length-1])  # Output sequences (only the last time step)

# Converting the sequences to numpy arrays
X0 = np.array(X_sequences)
Y0 = np.array(y_sequences)

# Normalize the input data with L2
X = X0 / np.linalg.norm(X0, ord=2, axis=0, keepdims=True)
Y = Y0 / np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

# Defining the sequential model
model = Sequential()

# Bidirectional LSTM Model
model.add(Bidirectional(LSTM(128, activation="tanh", return_sequences=True,
input_shape=(step, 2))))
model.add(Bidirectional(LSTM(128, activation="tanh", return_sequences=False))
model.add(Dropout(0.2)))
model.add(Dense(2, activation="linear"))

# Stop based on error
class StopTrainingOnMSE(Callback):
    def __init__(self, mse_threshold=0.0000003):
        super(StopTrainingOnMSE, self).__init__()
        self.mse_threshold = mse_threshold

    def on_epoch_end(self, epoch, logs=None):
        current_mse = logs.get('mse')  # Ensure 'mse' matches the metric used
        if current_mse is not None and current_mse < self.mse_threshold:
            self.model.stop_training = True
```

```
            print(f"Training stopped. MSE ({current_mse}) is below the threshold
({self.mse_threshold}).")

# Patience callback
class DecayLearningRate(tf.keras.callbacks.Callback):
    def __init__(self, factor=0.97, patience=100):
        super(DecayLearningRate, self).__init__()
        self.factor = factor
        self.patience = patience
        self.best_loss = np.inf
        self.wait = 0

    def on_epoch_end(self, epoch, logs=None):
        current_loss = logs.get('loss')
        if current_loss < self.best_loss:
            self.best_loss = current_loss
            self.wait = 0
        else:
            self.wait += 1
            if self.wait >= self.patience:
                lr = tf.keras.backend.get_value(self.model.optimizer.lr)
                tf.keras.backend.set_value(self.model.optimizer.lr, lr * self.factor)
                print("Learning rate reduced to:",
tf.keras.backend.get_value(self.model.optimizer.lr))
                self.wait = 0

# Apply the patience callback
callback = DecayLearningRate(factor=0.97, patience=100)

# Define the custom stopping criterion callback
stop_on_mse_callback = StopTrainingOnMSE(mse_threshold=0.0000003)

# Compiling the model
model.compile(optimizer=Adam(0.001), loss='mean_squared_error', metrics=['mse'])

# Training the model
history_callback = model.fit(X, Y, epochs=5000, batch_size=800, callbacks=[callback,
stop_on_mse_callback], validation_split=0.2)

# Saving the model to a .h5 file
model.save(f"model_BidirectionalLSTM{step}.h5")

loss, mse = model.evaluate(X, Y)

print("Loss:", loss)

# Predicting results
result = model.predict(X)
result2 = result * np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)
```

```
# Saving the results to an Excel file
df_result = pd.DataFrame(result2, columns=['Column1', 'Column2'])
df_result.to_excel('result.xlsx', index=False)

# Plotting the MSE metric over time
plt.plot(history_callback.history['mse'], label='MSE (Training)')
plt.plot(history_callback.history['val_mse'], label='MSE (Validation)')
plt.title('MSE across epochs')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

5. Machine Learning experiment in MLFLOW inside Gloogle Colab

```
!pip install mlflow
!pip install ngrok
!pip install pyngrok
!pip install keras

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout,LSTM, Bidirectional,GRU
from tensorflow.keras.optimizers import Adam, SGD, Adamax
from sklearn.preprocessing import Normalizer
import numpy as np
import pandas as pd
import time
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import Callback
import mlflow
import mlflow.tensorflow
import matplotlib.pyplot as plt

!mlflow server --host 0.0.0.0 --port 5000 --backend-store-uri sqlite:///mlflow.db --default-
artifact-root ./mlruns and> mlflow.log and

import subprocess

MLFLOW_TRACKING_URI = "sqlite:///mlflow.db"  # Substitua pelo diretório correto onde
deseja armazenar os registros do MLflow
# Inicie o servidor MLflow na porta 5000
subprocess.Popen(["mlflow", "ui", "--backend-store-uri", "sqlite:///mlflow.db", "--port",
"5000", "--host", "0.0.0.0", MLFLOW_TRACKING_URI])
mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)

!mkdir data
!mkdir models
```

```
# Commented out IPython magic to ensure Python compatibility.
# %%capture
# !wget -nc https://d37ci6vzurychx.cloudfront.net/trip-data/green_tripdata_2021-01.parquet -
P data
# !wget -nc https://d37ci6vzurychx.cloudfront.net/trip-data/green_tripdata_2021-02.parquet -
P data

"""*MODEL IN MLFLOW**"""
def train_dl(current_step, model_name, Bidirectional_name):
    mlflow.set_experiment("DLExperiment")

    with mlflow.start_run():
        mlflow.tensorflow.autolog()

        # Register tags
        mlflow.set_tag("model_name", model_name)
        mlflow.set_tag("Bid", Bidirectional_name)
        mlflow.set_tag("current_step", current_step)

        # Importing input data from Excel
        df_input = pd.read_excel('INPUT.xlsx')
        # Importing output data from Excel
        df_output = pd.read_excel('OUTPUT.xlsx')

        # Converting input data to a numpy array
        X_train = df_input.values
        # Converting output data to a numpy array
        y_train = df_output.values

        # Organizing data into sequences
        seq_length = current_step  # Sequence length
        num_samples = X_train.shape[0] - seq_length + 1

        X_sequences = []
        y_sequences = []

        for i in range(num_samples):
            X_sequences.append(X_train[i:i+seq_length])  # Input sequences
            y_sequences.append(y_train[i+seq_length-1])  # Output sequences (only the last time
step)

        # Converting sequences to numpy arrays
        X0 = np.array(X_sequences)
        Y0 = np.array(y_sequences)

        # Normalize the input data with L2
        X = X0 / np.linalg.norm(X0, ord=2, axis=0, keepdims=True)
        Y = Y0 / np.linalg.norm(Y0, ord=2, axis=0, keepdims=True)

        if Bidirectional_name == 4:  # WITHOUT BIDIRECTIONALITY
```

```
        model = Sequential()

        if model_name == 1:  # SIMPLER RNN

            model.add(SimpleRNN(128, activation="relu", return_sequences=True,
input_shape=(current_step, 3)))
            model.add(SimpleRNN(128, activation="relu", return_sequences=False))
            model.add(Dropout(0.2))
            model.add(Dense(5, activation="linear"))

        if model_name == 2:  # LSTM

            model.add(LSTM(128, return_sequences=True, input_shape=(current_step, 3)))
            model.add(LSTM(128, return_sequences=False))
            model.add(Dropout(0.2))
            model.add(Dense(5, activation="linear"))

    if Bidirectional_name == 3:  # WITH BIDIRECTIONALITY

        model = Sequential()

        if model_name == 1:  # SIMPLE RNN

            model.add(Bidirectional(SimpleRNN(128, activation="relu",
return_sequences=True, input_shape=(current_step, 3))))
            model.add(Bidirectional(SimpleRNN(128, activation="relu",
return_sequences=False)))
            model.add(Dropout(0.2))
            model.add(Dense(5, activation="linear"))

        if model_name == 2:  # LSTM

            model.add(Bidirectional(LSTM(128, return_sequences=True,
input_shape=(current_step, 3))))
            model.add(Bidirectional(LSTM(128, return_sequences=False)))
            model.add(Dropout(0.2))
            model.add(Dense(5, activation="linear"))

    # Stop based on error
    class StopTrainingOnMSE(Callback):
        def __init__(self, mse_threshold=0.000001):
            super(StopTrainingOnMSE, self).__init__()
            self.mse_threshold = mse_threshold

        def on_epoch_end(self, epoch, logs=None):
            current_mse = logs.get('mse')  # Make sure 'mse' matches the metric used
            if current_mse is not None and current_mse < self.mse_threshold:
                self.model.stop_training = True
```

```
                print(f"Training stopped. MSE ({current_mse}) is below the threshold
({self.mse_threshold}).")

        # Patience callback
        class DecayLearningRate(tf.keras.callbacks.Callback):
            def __init__(self, factor=0.97, patience=25):
                super(DecayLearningRate, self).__init__()
                self.factor = factor
                self.patience = patience
                self.best_loss = np.inf
                self.wait = 0

            def on_epoch_end(self, epoch, logs=None):
                current_loss = logs.get('loss')
                if current_loss < self.best_loss:
                    self.best_loss = current_loss
                    self.wait = 0
                else:
                    self.wait += 1
                    if self.wait >= self.patience:
                        lr = tf.keras.backend.get_value(self.model.optimizer.lr)
                        tf.keras.backend.set_value(self.model.optimizer.lr, lr * self.factor)
                        print("Learning rate reduced to:",
tf.keras.backend.get_value(self.model.optimizer.lr))
                        self.wait = 0

        # Apply the patience callback
        callback = DecayLearningRate(factor=0.97, patience=25)

        # Define the custom stopping criterion callback
        stop_on_mse_callback = StopTrainingOnMSE(mse_threshold=0.000001)

        # Compile the model
        model.compile(optimizer=Adam(0.01), loss='mean_squared_error', metrics=['mse'])

        history = model.fit(X, Y, epochs=500, batch_size=1500, callbacks=[callback,
stop_on_mse_callback], validation_split=0.2)

        # Save the model to a .h5 file
        model.save(f"{'SimpleRNN' if model_name == 1 else 'LSTM'}{'Bidirectional' if
Bidirectional_name == 3 else 'Simple'}{current_step}.h5")
        mlflow.log_artifact(f"{'SimpleRNN' if model_name == 1 else 'LSTM'}{'Bidirectional' if
Bidirectional_name == 3 else 'Simple'}{current_step}.h5")

        loss, mse = model.evaluate(X, Y)

        # Log metrics
        mlflow.log_metric("mse", mse)

        # Plot for errors and accuracy
```

```
        plt.plot(history.history['loss'])
        plt.savefig("loss.png")
        plt.plot(history.history['mse'])
        plt.savefig("mse.png")

        # Log artifacts
        mlflow.log_artifact("loss.png")
        mlflow.log_artifact("mse.png")

        mlflow.tensorflow.autolog()

        # Execution information
        print("Model: ", mlflow.active_run().info.run_uuid)
    mlflow.end_run()

"""**CREATING TUNNEL INSIDE GOOGLE COLAB TO ACCESS LOCALHOST
GRAPHICAL INTERFACE**"""

import mlflow
import subprocess
from pyngrok import ngrok, conf
import getpass

# Terminate open tunnels if they exist
ngrok.kill()
print("Enter your authtoken, which can be copied from https://dashboard.ngrok.com/auth")
conf.get_default().auth_token = getpass.getpass()
port = 5000
public_url = ngrok.connect(port).public_url
print(f' * ngrok tunnel "{public_url}" -> "http://127.0.0.1:{port}"')

"""**SIMULATING**"""

step = [10]
model_type = [1, 2]  # 1 for SimpleRNN and 2 for LSTM
Bidirectionallist = [3, 4]  # 3 for bidirectional and 4 for non-bidirectional

for current_step in step:
    for model_name in model_type:
        for Bidirectional_name in Bidirectionallist:
            train_dl(current_step, model_name, Bidirectional_name)
```

6. Python environment simulation

```
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import load_model
```

```
import time
import numpy as np
import random
import math
from scipy.interpolate import UnivariateSpline

# Context of the recurrent neural network
step = 10

######################### START SIMULATION PARAMETERS
############################

# Create DataFrame to store data
df = pd.DataFrame(columns=['Flowmass', 'Temperature'])
df2 = pd.DataFrame(columns=['K', 'TAU', 'THETA', 'kp', 'ki'])

# Load saved RNN models
model_rnn = load_model(f'model_LSTM{step}.h5')

# Load saved ANN models
model_ann = load_model('model_ANN.h5')

# Load input values
input = pd.read_excel('INPUT.xlsx')

# Load output values
output = pd.read_excel('OUTPUT.xlsx')

# Convert input data to a numpy array
X_train_RNN = input.values

# Convert output data to a numpy array
Y_train_RNN = output.values

# Convert input data to a numpy array
X_train_ANN = input.values

# Convert output data to a numpy array
Y_train_ANN = output.values

# Convert sequences to numpy arrays
X0A = np.array(X_train_ANN)
Y0A = np.array(Y_train_ANN)

# Normalize the input data with L2
XA = X0A / np.linalg.norm(X0A, ord=2, axis=0, keepdims=True)
YA = Y0A / np.linalg.norm(Y0A, ord=2, axis=0, keepdims=True)

# Convert sequences to numpy arrays
X1 = np.array(X_train_RNN)
```

```
Y1 = np.array(Y_train_RNN)

# Finding normalization variables with L2
X_RNN = X1 / np.linalg.norm(X1, ord=2, axis=0, keepdims=True)
Y_RNN = Y1 / np.linalg.norm(Y1, ord=2, axis=0, keepdims=True)

# Organizing data into sequences
seq_length = step  # Sequence length
num_samples = X_train_RNN.shape[0] - seq_length + 1

# Assuming X_RNN is your input data with shape (None, 2)
X_sequences_RNN = []

for i in range(X_RNN.shape[0] - seq_length + 1):
    X_sequences_RNN.append(X_RNN[i:i + seq_length])

X_input_RNN = np.array(X_sequences_RNN)

# Prediction from the RNN
output_rnn = model_rnn.predict(X_input_RNN) * np.linalg.norm(Y1, ord=2, axis=0,
keepdims=True)

# Prediction from the ANN
constants = model_ann.predict(XA) * np.linalg.norm(Y0A, ord=2, axis=0, keepdims=True)

# Define variables for the control loop
t = np.linspace(0, 8800, 8800)  # Simulation time
dt = t[1] - t[0]

e = np.zeros(len(t))  # Error
u = np.zeros(len(t))  # Control signal
y = np.zeros(len(t))  # Plant output
y_sp = np.full(len(t), 16)  # Setpoint

# Define initial conditions
y[0] = 26  # Ambient temperature

# List to store predictions
predictions = []

# Control loop
for i in range(1, 8800):

    e[i] = y_sp[i] - y[i - 1]

    # Define the first-order plant with time delay and gains

    K = constants[i, 0]
    tau = constants[i, 1]
    theta = constants[i, 2]
```

```
    Kp_c = output_rnn[0, 3]
    Ki_c = output_rnn[0, 4]

    # Calculate the control signal using the PID controller
    if i <= 1:
        u[i] = Kp_c * e[i] + Ki_c * e[i] * dt
    else:
        integral = Ki_c * np.sum(e[:i]) * dt
        proportional = Kp_c * e[i]
        u[i] = proportional + integral

    # Simulate the first-order plant with time delay
    if i < theta:
        y[i] = y[0]
    else:
        y[i] = (np.exp(-(i - theta) / tau) * y[0] +
                (1 - np.exp(-(i - theta) / tau)) * K * (u[i]))

    # Add the calculated variables to the predictions list
    predictions.append([K, tau, theta, Kp_c, Ki_c])

    # Add the current temperature to the DataFrame
    df = pd.concat([df, pd.DataFrame({'Time': [t[i]], 'Error': [e[i]], 'Temperature': [y[i]],
'ControlSignal': [u[i]]})], ignore_index=True)

# Save the DataFrame to an Excel file
df.to_excel(f'Temperature_LSTM{step}.xlsx', index=False)

# Create a Pandas DataFrame with predictions
dfp = pd.DataFrame(predictions, columns=['K', 'tau', 'theta', 'kp', 'ki'])

# Save the predictions to an Excel file
dfp.to_excel(f'predictions_LSTM{step}.xlsx', index=False)

# Plot the results
plt.plot(t, y, label='Temperature')
plt.plot(t, u, label='ControlSignal')
plt.plot(t, y_sp, label='Setpoint')
plt.legend()
plt.xlabel('Time [s]')
plt.ylabel('Temperature [ºC]')
plt.xlim(xmin=0)
plt.xlim(xmax=8000)
plt.ylim(ymin=0)
plt.ylim(ymax=40)
plt.show()
```

7. Raspberry pi code

```
import board
import RPi.GPIO as GPIO
import numpy as np
import digitalio
import adafruit_max31855
import time
import csv
import requests  # Import the requests library

# Function to write the temperature to a CSV file
def write_to_csv(temperature):
    with open('temperatures.csv', mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([time.strftime("%Y-%m-%d %H:%M:%S"), temperature])

# Function to set up the temperature sensor
def setup_max31855():
    spi = board.SPI()
    cs = digitalio.DigitalInOut(board.D8)  # Chip Select pin
    return adafruit_max31855.MAX31855(spi, cs)

# Function to get the controller gains via GET
def get_controller_gains():
    response = requests.get('http://54.173.246.38:8000/api/v1/doutorado/latest')
    data = response.json()
    return data['kp'], data['ki']

# Create the CSV file header
with open('temperatures.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Timestamp', 'Temperature (C)'])

###### PWM PIN CONFIGURATION #####

# Define the pin to be used
pin = 12

# Define the PWM signal frequency
freq = 60

# Initial GPIO setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(pin, GPIO.OUT)

# Initialize the PWM object with the specified frequency
pwm = GPIO.PWM(pin, freq)

# Start the PWM signal with a 0% duty cycle
pwm.start(0)
```

```
# Setpoint
setpoint = 14

# Buffer to store the last three temperature readings
temp_buffer = [0]

# Get the controller gains via GET
Kp_c, Ki_c = get_controller_gains()
Kd_c = 0  # Derivative gain of the controller

# Define variables for the control loop
t = np.linspace(0, 800000, 800000)  # Simulation time
dt = t[1] - t[0]  # Time interval

# Define 'e' before the loop
erro = np.zeros(len(t))  # Error
u = np.zeros(len(t))  # Control signal
y_sp = np.full(len(t), setpoint)  # Setpoint

# Counter
j = 0

while True:
    # Update the controller gains via GET
    Kp_c, Ki_c = get_controller_gains()

    time.sleep(0.33)

    try:
        # Initialize the MAX31855 object
        max31855 = setup_max31855()

        # Read the temperature in Celsius
        if len(temp_buffer) >= 3:
            temp_buffer.pop(0)
        temp_buffer.append(float(max31855.temperature))
        temp = np.mean(temp_buffer)

        # Calculate the error
        erro[j] = y_sp[j] - temp
        print("Current error:", erro[j])

        # Display the temperature
        print("Temperature: {:.2f} °C".format(temp))

        # Write the temperature to the CSV file
        write_to_csv(temp)

        # Accumulated error
```

```
        acumulo_de_erro = np.sum(erro[:j])

        # Check if the error sum exceeded 220
        if acumulo_de_erro > 220:
            acumulo_de_erro = 220
        elif acumulo_de_erro < -220:
            acumulo_de_erro = -220

        # Calculate the control signal using the PID controller
        if j == 1:
            u[j] = Kp_c * erro[j] + Ki_c * erro[j] * dt + Kd_c * 0
        else:
            u[j] = Kp_c * erro[j] + Ki_c * acumulo_de_erro * dt + Kd_c * (erro[j] - erro[j-1]) / dt

        # Adjust the duty cycle
        if u[j] > 1:
            u[j] = 1
        if u[j] < 0:
            u[j] = 0

        u[j] = u[j] * 100

        if u[j] > 100:
            u[j] = 100

        pwm.ChangeDutyCycle(u[j])
        print("Control signal", u[j])
        print("Variable J:", j)
        print("Error sum:", acumulo_de_erro)

        j += 1

    except KeyboardInterrupt:
        # End the program when the user presses Ctrl+C
        print("Program terminated by user")
        break

    except Exception as ex:
        # Catch any other exceptions and do nothing
        print("Exception: " + str(ex))
        pass
```

8. Docker commands, Terraform e AWS CLI

```
#######COMANDOS DOCKER############

##order for make containers ###
primeiro database
depois front end
```

depois api

#start one container
docker start namecontainer

start all containers
docker start $(docker ps -aq)

restart all containers
docker restart $(docker ps -aq)

#Admin power
sudo chown -R ramon /home/ramon/APIDOUTORADO

#Open VSCODE in linux
code .

#make imagem with docker-compose
docker-compose build

docker exec -it 63b56571f634 nc -zv 172.17.0.2 5432
comando para testar conexao

stop all containers
docker stop $(docker ps -q)

remove all containers
docker rm $(docker ps -a -q)

romove all images
docker rmi $(docker images -q)

#make containers
docker run -p 5432:5432 -d --name database apidoutorado_database

#get ip number of the container
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
f591cdade92d

#inlet in container database
docker exec -it database psql -U ramon -d ramon2

#create table in container database
SQL Command

#insert data in container database
SQL Command

shows data in container database
SQL Command

```
#out of the container database
\q

#make tags for send images for Dockerhub
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
docker push ramondepaoli/thermalcontrolml:apidoutorado_api

########## CONTAINERS IN AWS###############################

#Install client AWS
sudo snap install aws-cli --classic
aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]: json

# ssh conection
ssh -i "Doutorado1.pem" ubuntu@ec2-54-94-26-162.sa-east-1.compute.amazonaws.com

# Docker install in EC2
sudo snap install docker

#tag nas imagens
docker tag mlapply_api depaoli91/thermalcontrol:mlapply_api

# send images for docker hub
docker push depaoli91/thermalcontrol:mlapply_api

#dowload images in EC2-AWS
docker pull depaoli91/thermalcontrol:mlapply_api
```