Design and Application of Physical Contextual Computers

by

Vahideh Shirmohammadli

M.Sc., University of Guilan, 2015

B.Sc., University of Guilan, 2013

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Mechatronic Systems Engineering
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY

Spring 2024

Declaration of Committee

Name:	**Vahideh Shirmohammadli**
Degree:	**Doctor of Philosophy**
Title:	**Design and Application of Physical Contextual Computers**
Committee:	**Chair: Carolyn Sparrey** Associate Professor, Mechatronic Systems Engineering
	Behraad Bahreyni Supervisor Professor, Mechatronic Systems Engineering
	Ljiljana Trajkovic Committee Member Professor, Engineering Science
	Mo Chen Committee Member Assistant Professor, Computing Science
	Gary Wang Examiner Professor, Mechatronic Systems Engineering
	Shervin Shirmohammadi External Examiner Professor, Electrical Engineering and Computer Science University of Ottawa

Abstract

The widespread adoption of Internet of Things (IoT) devices has ushered in an era of extensive sensor-generated data, leading to the need for improved communication capabilities, data storage, and energy efficiency. A significant proportion of the power used by these devices, ranging from 60% to 98%, is dedicated to energy dissipation through communication lines. This thesis addresses these requirements through innovative computing approaches that center on processing data in close proximity to sensors.

A key contribution of this research is the introduction of "thermo-computing," a ground breaking concept that uses the unique characteristics of materials and devices to process data over time. Thermo-computing leverages an entirely passive network of thermistors for data manipulation, promising significant energy savings. This work, supported by extensive experiments and careful analysis, firmly establishes thermo-computing as a transformative method for data processing.

Additionally, the thesis examines the capabilities of 3D-printed computing platforms for real-time data processing and the classification of sensory data. It investigates how memory, nonlinearity, and sampling rates affect the performance of these processors, highlighting their cost-effectiveness and ease of integration into existing smart and 3D-printed intelligent systems, marking a noteworthy advancement in the domain of 3D-printed systems.

Furthermore, the research bridges the gap between software-based reservoir computing and its physical counterpart, introducing a novel reservoir computing platform. This pioneering platform enhances our understanding of the applications of physical reservoir computing and explores the transfer of energy within physical systems acting as reservoirs inspired by neural networks. This innovative approach has the potential to transform physical computing platforms and offer new solutions across areas such as networks, sensing, and computing.

Keywords: reservoir computing; physical reservoir computing; energy transfer; thermo-reservoir; 3D-printed reservoir

Publications

- V. Shirmohammadli and B. Bahreyni, “Role of Heat Transfer in Shaping the Dynamics of Physical Reservoir Computers,” *Journal of Physics D: Applied Physics*, Submitted.

- V. Shirmohammadli and B. Bahreyni, “A 3D-Printed Computer”, *Advanced Intelligent Systems*, vol. 5, p. 2300015, 2023.

- V. Shirmohammadli and B. Bahreyni, “A Neuromorphic Electrothermal Processor for Near-Sensor Computing,” *Advanced Material Technologies*, vol. 7, no. 11, p. 2200361, 2022.

- V. Shirmohammadli and B. Bahreyni, “Machine Learning for Sensing Applications: A Tutorial,” *IEEE Sensors Journal*, vol. 22, no. 11, pp. 10183 - 10195, 2022.

- V. Shirmohammadli and B. Bahreyni, “Development of a Thermo-Computing Platform,” *In Proc. 21st Int. Conf. on Solid-State Sensors, Actuators and Microsystems (Transducers)*, Orlando, FL, USA, June 2021, pp. 1307-1310.

Dedication

I dedicate this thesis to the journey of personal growth, with gratitude for the valuable lessons learned and the profound changes experienced along the way. May it serve as a reminder that education is not just about acquiring knowledge but also about becoming a better version of oneself.

Acknowledgements

I wish to express my deepest gratitude to the exceptional individuals who supported and guided me throughout this transformative experience.

First and foremost, my sincere appreciation goes to my supervisor, Prof. Behraad Bahreyni, who, without a doubt, was the best mentor I had ever. His generous sharing of extensive academic knowledge played a pivotal role in helping me establish myself within the academic community. His firm yet compassionate guidance not only facilitated significant contributions to publications during my PhD but also opened doors I never thought possible.

I also extend my gratitude to my esteemed PhD thesis committee members, Prof. Ljiljana Trajković and Prof. Mo Chen. Their dedicated time, valuable feedback, and thought-provoking questions have significantly enriched the quality of my work.

Special appreciation is reserved for my colleagues and peers at the School of Mechatronic Systems Engineering. Engaging in extracurricular activities and maintaining personal connections with all of you has been truly enriching.

Lastly, I would like to extend my heartfelt appreciation to my family for their enduring belief in my abilities, which has been a cornerstone of my academic accomplishments. To my spouse, you have been my rock and the guiding light that helped me navigate the challenges of this academic endeavour. Your enduring support, unwavering belief, and consistent encouragement have been sources of inspiration and motivation. I am deeply grateful for your presence by my side, which made this journey possible.

This thesis represents a collective effort, and I offer my sincere thanks to all those who have contributed to its successful completion. Your support and encouragement have played pivotal roles in making this achievement possible.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ANN	Artificial Neural Network
ASIC	Application-Specific Integrated Circuits
CPU	Central Processing Units
DDE	Delay Differential Equation
ESN	Echo State Network
ESP	Echo State Property
FNN	Feedforward Neural Network
FPGA	Field-Programmable Gate Array
GPU	Graphical Processing Units
LSM	Liquid State Machine
MC	Memory Capacity
ML	Machine Learning
MSE	Mean Squared Error
NMSE	Normalized Mean Squared Error
NRC	Neuromorphic Computing
NRMSE	Normalized Root Mean Squared Error
PRC	Physical Reservoir Computing
R^2	R-squared
RBF	Radial Basis Function
RC	Reservoir Computing
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SDN	Single Dynamical Node
SOI	Silicon-on-Insulator
SVM	Support Vector Machine
XOR	Exclusively-OR

Glossary

Dynamic System	In physics, an object or ensemble of objects whose state varies over time and obeys differential equations with time derivatives is called a dynamic system.
Edge of Chaos	An operating point for a system where complex patterns and behaviour emerge in a homogeneous medium is found.
Fading Memory	When a system state contains all information from previous time steps in a discrete-time domain, the most recent inputs receive the highest weight.
Injective Function	In mathematics, an injective function is a function g that maps distinct elements to another distinct elements; therefore, $g(x_1) = g(x_2)$ implies $x_1 = x_2$.
Kernel	A function that transforms the input values into a high-dimensional vector is referred to as a kernel.
Nonlinear Node/Neuron	Node and neuron are interchangeably used throughout this document. A (physical) node is a node/neuron that reacts nonlinearly to what it receives as its input (either physical, chemical, or electrical stimulation).
Non-temporal Task	In a non-temporal task, the individual data points are considered independent of each other, meaning that their order or timing does not affect the outcome of the task.
Reservoir	A network of randomly connected nonlinear nodes or elements leveraging a short memory of past information.
Spatial Reservoir	A reservoir in which the internal nonlinear nodes are spatially distributed.
Temporal Task	The desired output function may contain a memory of previous input values, where both the input and output signals are in the discrete-time domain.
Timescale	A representative of the response time of a nonlinear dynamical element. If data are introduced to and read from the same element at the rate of 1/timescale, the element will almost pass its dynamic region.
Timestep	A unit for representation of steps in a discrete-time domain. In the physical world, it is used to introduce an input value to a system for a determined duration of time.
Virtual Nodes	The internal nonlinear nodes are multiplexed in time in a specific reservoir type. These reservoir nodes are called virtual nodes.

Chapter 1.

Introduction

> *"Reservoir computing is a radically new way of doing computation that is based on the idea of using a dynamic system to perform a computation, rather than using a static set of weights or a set of rules."*
>
> *— Dr. John Paul Gosling, University of Oxford*

Technological advancements have enabled us to generate and record vast data using microsensors that can quantify real-world parameters. However, our ability to make sense of this data has not kept up with the pace of data generation, leading to bottlenecks in data processing and analysis that lead to delays in processing data, increased processing time, and even data loss or corruption. In many cases, this bottleneck can be addressed by limiting the storage and transmission of data to contextual information instead of raw data.

Typically, raw data must be transmitted and processed at the edge or cloud levels to generate context, as shown in Figure 1.1 [1]. This approach is wasteful in terms of communication bandwidth and data storage and accounts for a significant amount of energy usage in the Internet of Things (IoT) [2]. By limiting the data storage and transmission to contextual information rather than raw data, as shown in Figure 1.2, it is possible to achieve significant energy savings (up to an order of magnitude) and reduce latencies in time-sensitive applications. Context-based triggers can also significantly reduce power consumption in ultra-low-power systems by allowing the system to wake up only when needed to take measurements.

To address these challenges, researchers are exploring various computing approaches that operate near or inside sensory systems [3]. One of these approaches is reservoir computing, a recent trend in machine learning suitable for processing temporal data, such as time series, speech, and video [4]. Reservoir computing is a machine learning approach practical for various tasks, including pattern recognition, time series prediction, and control [4]–[8]. Unlike traditional machine learning algorithms requiring extensive training and data preprocessing, reservoir computing systems can learn and

adapt to new inputs online. One key aspect of reservoir computing is using a dynamic system, known as the "reservoir," to transform and scale input signals.

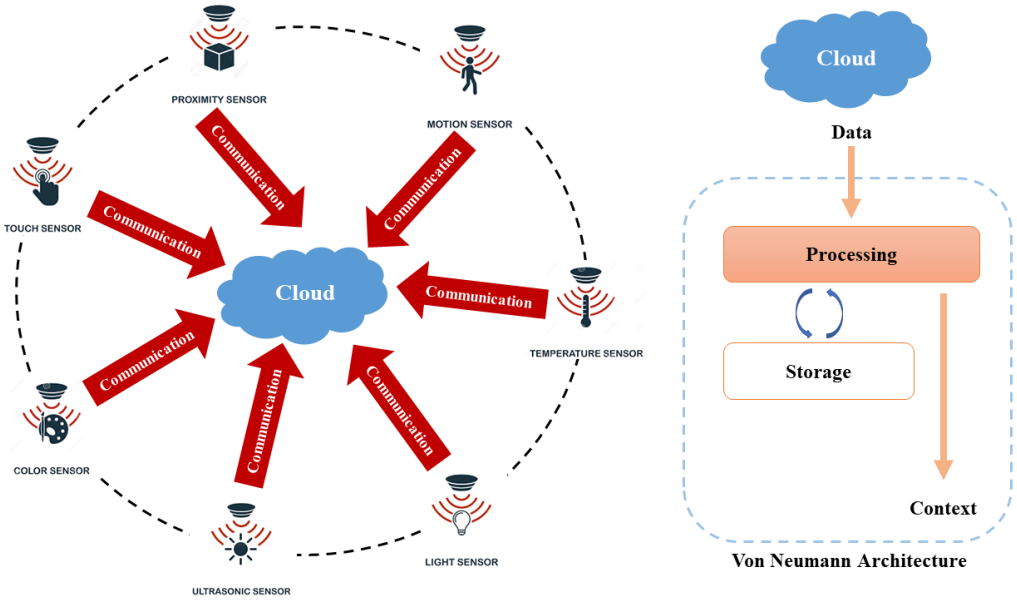


Figure 1.1. Conventional context detection in the internet of things (IoTs), where the processing is conducted in clouds. The cloud structure is generally based on a Von Neumann architecture.

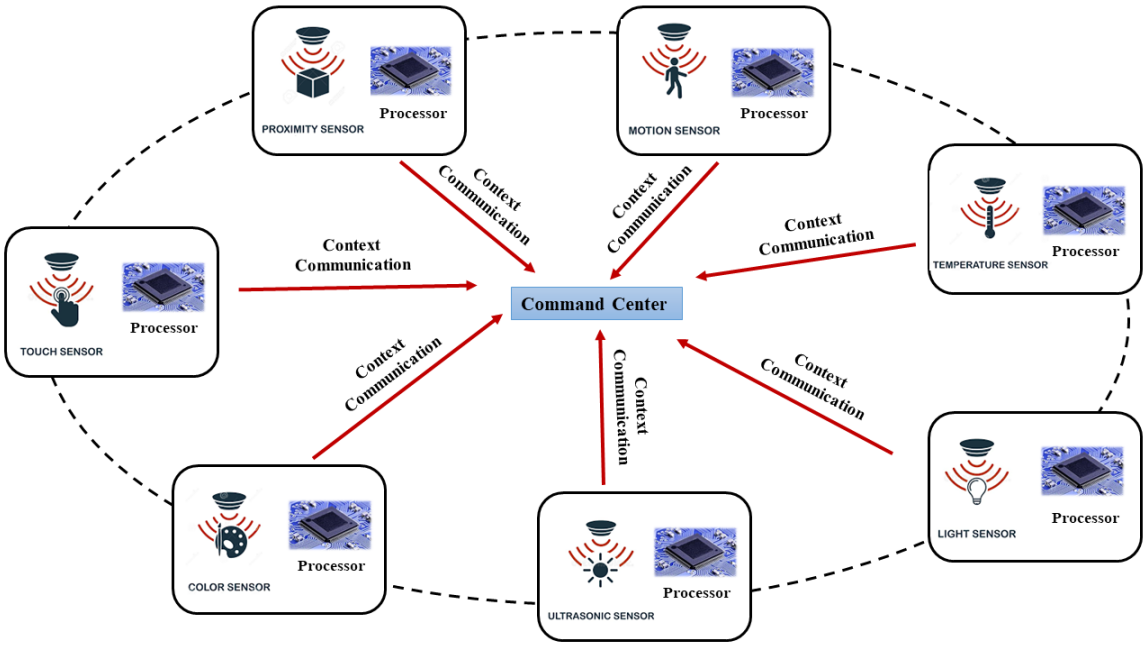


Figure 1.2. Unconventional context detection platform: moving processors close to the sensors to generate and communicate the context to command centers (near-sensor processing platform).

Over the past decade, there has been growing interest in developing physical reservoir computing platforms, which use physical systems, such as lasers, oscillators, and microfluidic channels, as the reservoir [9]–[12]. These physical platforms offer several potential benefits over traditional computing approaches, including increased energy efficiency, robustness to noise and perturbations, and the ability to perform computations at the nanoscale.

However, the development of physical reservoir computing platforms is a challenging task, as it requires a deep understanding of the complex dynamics of the physical system, as well as the ability to control and manipulate these dynamics in a precise and reliable manner.

This research aims to advance state-of-the-art physical reservoir computing platforms and enable the development of new and innovative computing applications. The goal of this project is to contribute to the development of physical reservoir computing platforms by exploring new approaches to the design, control, and optimization of these systems. In particular, this thesis focuses on developing 3D-printed reservoirs, which are promising physical systems for many processing applications.

1.1. Addressing the Data Processing Bottleneck

Data processing bottlenecks occur when the rate at which data are generated or received exceeds the rate at which it can be processed or stored. It can be due to various factors, including limited processing power, slow network connectivity, or a high volume of data. Researchers have developed different techniques to address these bottlenecks, depending on the root cause of the problem.

One effective solution to address a data processing bottleneck caused by limited processing power is to upgrade the hardware. This can involve several approaches, such as adding more CPUs, increasing memory, or upgrading to faster storage devices. Another approach is to leverage specialized processors that are optimized for specific processing tasks, such as graphics processing units (GPUs), field-programmable gate arrays (FPGAs), or application-specific integrated circuits (ASICs). Central processing units (CPUs) are the primary processors in a computer and are responsible for executing most of the instructions that run on a computer. These processors are designed to perform

general-purpose operations but may not be optimized for specific tasks. In contrast, GPUs are specialized processors that are designed to accelerate graphics and video processing tasks. These processors have many small, efficient cores optimized for parallel processing, making them well-suited for tasks involving large amounts of data. ASICs are custom-designed processors built to perform specific tasks and are used in applications that require high performance and low power consumption. These processors are highly optimized for their intended tasks and often outperform general-purpose CPUs. FPGAs are programmable processors that can be configured to perform a wide range of functions. They offer flexibility and reconfigurability, which makes them well-suited for applications that require rapid prototyping and development.

Distributed processing is another technique used to address data processing bottlenecks. It involves breaking up the data processing tasks into smaller, more manageable ones and distributing them across multiple processors or nodes. This technique can improve performance and reduce processing time by allowing multiple processors to work on different parts of the same task simultaneously [13].

If the bottleneck is caused by slow network connectivity, increasing the network bandwidth can help. It can involve upgrading network hardware, such as routers and switches, or using techniques such as load balancing to distribute traffic across multiple network links [14].

These solutions use the general von Neumann architecture, which has several limitations that impact the performance of computers. The von Neumann architecture is a computer design paradigm proposed by mathematician and computer scientist John von Neumann in the 1940s [15]. One of the main limitations of the von Neumann architecture is the separation of memory and processing units. In this architecture, the CPU fetches instructions from memory, decodes them, and executes them one at a time. It requires the CPU to constantly access the memory to fetch instructions, which can be slow and inefficient. It is known as the von Neumann bottleneck.

Many modern computers use caching, pipelining, and parallel processing techniques to improve performance to overcome the von Neumann bottleneck. However, these techniques can only do so much to overcome the fundamental limitations of the architecture. In this regard, researchers are exploring alternative architectures, such as

neuromorphic computing, analog computing based on Hebbian Learning, and reservoir computing. Neuromorphic computing is inspired by the human brain's structure and function. It involves the development of hardware and software systems that mimic the neural architecture and processes of the brain to perform tasks such as perception, learning, and decision-making. Neuromorphic computing systems are designed to be highly energy-efficient and capable of processing large amounts of data in real-time [16].

One example of neuromorphic computing is spiking neural networks, which use pulses or "spikes" to encode and transmit information, similar to how neurons in the brain communicate. Neuromorphic chips are specialized microprocessors designed to implement spiking neural networks or other types of neural architectures in hardware. For example, Intel introduced Loihi, a neuromorphic processor with on-chip learning [17]. IBM has also developed the neuromorphic TrueNorth chip with 70mW power consumption when implementing artificial neural networks for pattern recognition [18]. These chips can perform certain types of calculations much more efficiently than traditional CPUs, making them suitable for pattern recognition and classification tasks.

While neuromorphic computing shows promise for overcoming the limitations of von Neumann's architecture, its development and implementation still have challenges and limitations. Neuromorphic computing systems are often highly complex and can be challenging to design and implement. Programming neuromorphic computing systems can also be complicated due to their complexity and the lack of standardized approaches and tools. Moreover, neuromorphic computing systems can be expensive to develop and implement [16].

On the other hand, around the 1990s feedforward neural networks (FNNs) through analog components were implemented, which were able to harness the continuous nature of analog signals, aligning more closely with the principles of neural computation [19]. Researchers have explored diverse architectures to instantiate FNNs, leveraging the inherent parallelism and efficiency of analog circuits [20]. Analog implementations of FNNs typically involve neurons modelled with analog components, such as transistors, to simulate the dynamic behaviour of biological neurons. Synaptic connections, essential for weight adjustments, were often realized through tunable conductance.

An alternative computing approach called reservoir computing (RC) emerged in 2001 [5], [7], [21], [22]. RC is a machine learning algorithm designed to address the complexities in training recurrent neural networks (RNNs). It uses a fixed network of coupled nonlinear elements with a finite memory of past events, called a reservoir, and a trainable output layer, known as the readout. The reservoir maps sequential input data to a multi-dimensional state space, and the readout generates context from raw data using simple linear regression. RC offers temporal data processing capabilities and can go beyond the von Neumann architecture due to its short-term memory and computation combination.

RC has been successfully applied to various nonlinear time-series applications, such as sequential pattern classification, time series prediction, adaptive filtering and control, and feature extraction on complex time series [23]. It offers low training costs, real-time processing with high accuracy, and simple training algorithms, often using linear regression or generalized ridge regression. It has led to the development of physical reservoir computing, in which proposed physical systems leverage materials or devices' specific and nonlinear responses to perform well-defined tasks in the physical domain. Once a reservoir is designed, it can perform well on multiple tasks by training the output layer.

The randomly connected and temporally internal connections in a reservoir benefit the hardware implementation of next-generation computing platforms. This has led to recent developments in physical RC systems based on connected mechanical oscillators [12], [24], [25], memristors [26]–[32], photonic circuits [33]–[40], carbon nanotube/polymer composites [41]–[43], soft body [44], and connected atomic switches [11]. For example, bio-computers were developed at the organism or molecular level to interface with and respond to stimuli such as light and chemicals [45], [46]. In addition, reaction-diffusion computers use chemical reactions to perform highly parallel computations [47].

In a physical RC system, the nonlinearities of the materials or devices provide the mapping function, the system's transient response creates a fading memory which makes the network output independent of events in the distant past, and the fixed coupling and internal connections give the system a time-dependent response. For a reservoir to perform well in computational tasks, it must meet specific requirements [23]:

- The reservoir must exhibit nonlinearity, which is essential for tasks that are not linearly separable and for extracting nonlinear dependencies in time-series predictions.
- The reservoir must have fading memory, which is necessary to keep a short enough memory of past inputs to the system and is essential for temporal pattern recognition.
- The reservoir must have high dimensionality to map inputs into a high-dimensional state space and facilitate the separation of originally inseparable inputs and the retrieval of temporal dependencies of inputs.
- The reservoir must have a separation property, which ensures that the reservoir responses to different signals are sufficiently different, separating them into different classes.
- The reservoir must have an approximation property, which refers to the consistency of the reservoir and promotes the ability to be insensitive to small fluctuations such as noise.

1.2. Motivation

The development of physical RC platforms in this thesis has been motivated by the need for efficient and practical approaches to processing and analyzing data generated by sensory networks. As the amount of data generated by these networks continues to grow, there is a need for computing approaches that can handle this data more efficiently and effectively. RC platforms offer several benefits, including low training cost, real-time processing, and high accuracy using *simple training algorithms*.

Another motivation for developing physical RC platforms is the potential for *improved energy efficiency*. Traditional von Neumann computing architectures rely on data transfer between memory and processing units, which can be energy-intensive. In contrast, RC platforms can operate in a more distributed, local manner, with computation integrated within each sensory node throughout the network. This approach can lead to improved energy efficiency, particularly in the case of large-scale sensory networks.

Physical RC platforms also offer the potential for *increased flexibility and adaptability*. Training the output layer of an RC network makes it possible to adapt the network to perform different tasks without redesigning the entire system. This attribute can benefit applications where the data or task requirements may change over time.

One of the intrinsic motivations for my thesis is the *development of near-sensor processors for integrating sensors and processors*. We aim to help advance the knowledge in this field and pave the way for in-sensor processing. While there have been advancements in the development of physical computing platforms from various materials and devices, there is currently a lack of processors specifically designed for the compatible integration of sensors and processors. In this regard, there are many open questions and challenges in the development of these platforms, including:

- How can physical RC platforms be designed and implemented for near-sensor computing applications?
- What are the key challenges and limitations in developing physical RC platforms for near-sensor computing?
- How does the performance of physical RC platforms compare to traditional computing approaches?
- Can physical RC platforms be adapted to perform various tasks, or are they limited to specific tasks?

Specifically, I demonstrate the following:

- Design and implement physical RC platforms for near-sensor computing applications.
- Assess the feasibility and potential of physical RC for near-sensor computing applications.
- Evaluate the performance of physical RC platforms compared to traditional computing approaches.
- Determine the adaptability of physical RC platforms to perform a range of tasks.
- Identify and address the key challenges and limitations in developing physical RC for near-sensor computing.
- Provide recommendations for the design and development of physical RC for near- or in-sensor computing applications based on the findings of this research.

1.3. Structure

This thesis is organized as follows:

- Chapter 2 elaborates on the theory of reservoir computing and reviews the literature on physical reservoir computing platforms.

- Chapter 3 presents the initial efforts to identify materials and devices with specific properties for use in physical RC and the design of physical neurons. It also includes descriptions of our prototypes using off-the-shelf components and the implementation of basic information transfer mechanisms within the proposed reservoirs.

- Chapter 4 focuses on exploring printable materials with specific properties, developing information transfer mechanisms using thermal energy transfer, and the 3D printing of proposed processors.

- Chapter 5 evaluates and optimizes the performance of the 3D-printed processors.

- Chapter 6 discusses the future direction of the research and potential applications for physical RC.

Chapter 2.

Physical RC Systems

Physical reservoir computing is a form of machine learning that utilizes the dynamics of a physical system as the "reservoir" to perform computations. The basic idea behind physical RC is to use the intrinsic dynamics of the physical system to map input data into a high-dimensional feature space, where a simple linear model can be trained to produce the desired output [23]–[25].

Any dynamic system, such as electronic circuits, lasers, mechanical systems, or fluid, can serve as the physical system in RC [51]. The choice of the system is based on the application's specific requirements, including factors like speed, accuracy, and energy consumption. To process the input data, it is fed into the chosen physical system as a time-varying signal. The response of the system to the input signal is then recorded through an interface circuitry and analyzed to extract relevant features. These features are then used to train a linear model, also known as the "readout layer," which maps the features to the desired output. The readout layer is trained using a supervised learning algorithm like linear or ridge regression. The training process aims to find the optimal set of readout layer weights that minimize the error between the predicted and actual output.

Physical RC has been the subject of extensive research over the past decade, and numerous studies have investigated its theoretical properties and practical applications [52]. One of the key advantages of physical RC is that it can be implemented using low-power, analog hardware, which makes it well-suited for applications in embedded systems and the IoT. Moreover, physical RC has been shown to be effective in various applications, including speech recognition, image processing, and control of nonlinear systems. Physical RC is a promising machine learning approach that harnesses physical systems' dynamics to perform computations. Physical RC is specifically promising for developing the next generation of processors that can be located close to or inside sensory networks, offering energy-efficient and real-time information processing [3]. Here a brief literature study on the recently developed physical RC using various physical phenomena utilized for the reservoir is provided.

2.1. Statistical Learning for Time-series Data

Neural networks are a class of machine learning models that can be used for data analysis. In a feedforward neural network, as shown in Figure 2.1 (a), information flows in one direction, from the input layer through one or more hidden layers to the output layer. The input to a feedforward neural network can be represented as a sequence of samples, n. The weights and biases in the network are learned through a process called backpropagation, which involves minimizing a loss function that measures the discrepancy between the network's predictions and the actual values. The forward pass of a neural network can be represented as follows [53]

$$\begin{aligned} \boldsymbol{x}(n) &= \boldsymbol{f}(W\,\boldsymbol{u}(n) + b) \\ \boldsymbol{y}(n) &= W_{out}\,\boldsymbol{x}(n) \end{aligned} \tag{2-1}$$

where $\boldsymbol{x}(n)$ represents the hidden state at step n, $\boldsymbol{u}(n)$ represents the input at step n, W represents the weight matrix connecting the input to the hidden layer, $\boldsymbol{b}$ represents the bias vector for the hidden layer, f is a nonlinear activation function applied element-wise to the hidden state, W_{out} represents the weight matrix connecting the hidden layer to the output layer, $\boldsymbol{y}(n)$ represents the predicted output at step n. The weight matrices W and W_{out} are learned during the training process to optimize the network's performance on the given time-series data.

While NNs can effectively capture complex patterns in data, they lack the ability to explicitly model temporal dependencies. This is where recurrent neural networks (RNNs) excel [53]. By introducing recurrent connections, as shown in Figure 2.1 (b), RNNs can maintain an internal memory or hidden state that retains information about past inputs, allowing them to capture long-term dependencies in sequential data. This memory allows RNNs to process time-series data more effectively than traditional feedforward NNs. The recurrent connections in RNNs enable them to dynamically adapt their hidden state based on both current and previous inputs, making them well-suited for tasks that involve sequential information, such as speech recognition. The hidden state of an RNN at each time step can be calculated using the following formula

$$\boldsymbol{x}(n) = \boldsymbol{f}(\boldsymbol{W}_{hh}\,\boldsymbol{x}(n-1) + \boldsymbol{W}_{\boldsymbol{in}}\,\boldsymbol{u}(n) + b_h) \tag{2-2}$$

where ***x**(n)* represents the hidden state at timestep *n*, ***u**(n)* represents the input at timestep *n*, W_{hh} represents the weight matrix connecting the previous hidden state to the current hidden state, W_{in} represents the weight matrix connecting the input to the current hidden state, b_h represents the bias vector for the hidden state, and *f* is a nonlinear activation function applied element-wise to the hidden state. The output of an RNN at each time step can be obtained by multiplying the hidden state by an output weight matrix and applying an activation function.

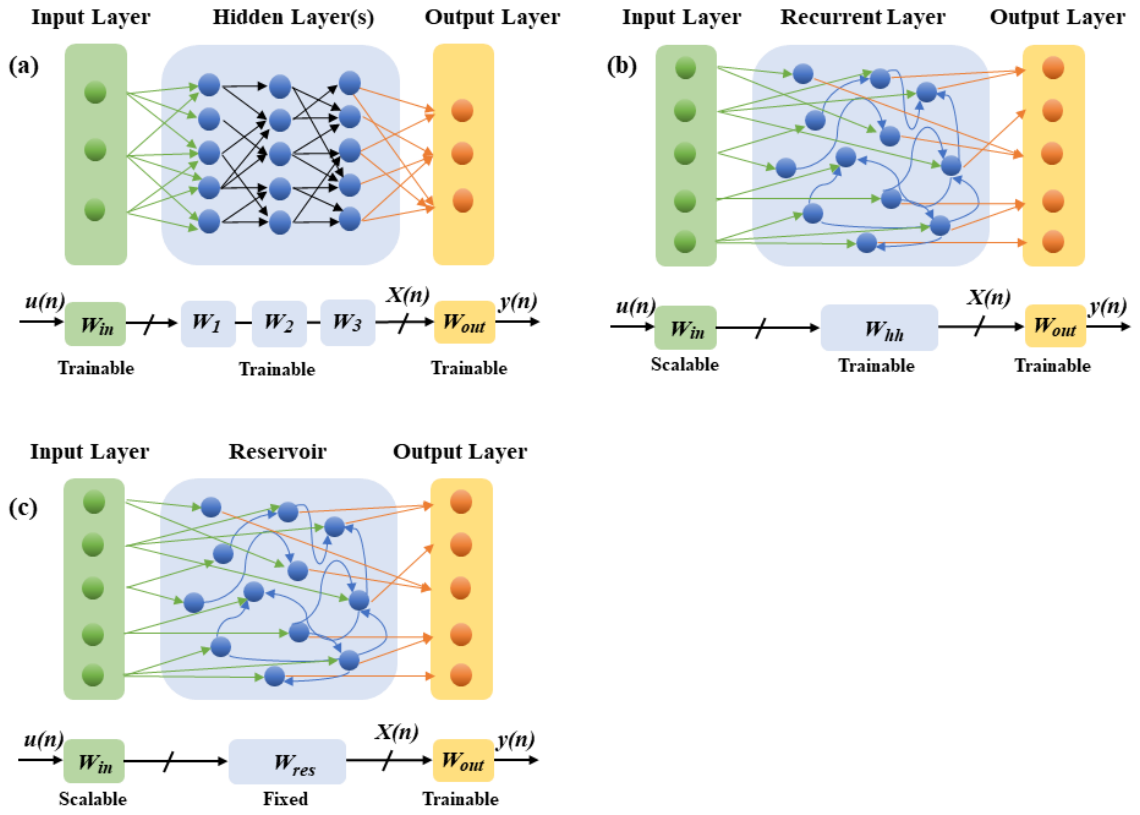


Figure 2.1. A graphic representation of (a) a neural network, (b) recurrent neural network, and (c) a reservoir computer. The reservoir has fixed nodes and connections. Only the weights at the readout layer require updates for specific data at the input.

2.2. Reservoir Computing

Recurrent neural networks have been used for temporal pattern recognition since the 1980s. Internal connections inside a conventional recurrent network require training,

a complex, time-consuming problem to solve [54], [55]. To tackle this issue, Jaeger introduced echo state networks (ESNs) in 2001, which focused on training dynamical systems for temporal learning tasks using RNNs [56]. Alternatively, Maass et al., proposed liquid state machines (LSMs) in 2002 for realistic modelling of the computational properties of neural microcircuits [57]. These solutions rely on a fixed, connected network of specific nodes (*i.e.*, fixed connection weights). Later, reservoir computing was coined by Verstraeten et al. in 2007 [22] to include these two approaches for implementing and training recurrent neural networks. RC successfully reduces the training of RNNs to a simple linear regression problem. Compared to a typical neural network, training is conducted only on the output layer with configurable weights. RC structure is compared to a typical neural network in Figure 2.1 (c). In general, RC consists of three parts [54]:

- An input layer for scaling and introducing the input to the reservoir: The input layer maps the input signal to the nonlinear functions in the reservoir with a bias.
- The reservoir of connected nonlinear functions with fixed internal weights: This reservoir structure remains unchanged for all applications.
- The output or readout layer is the trainable layer, in which a linear weighted sum of the input-excited reservoir states creates the predicted/classified output. Usually, linear regression is utilized to update and optimize the weights of this linear combination by introducing the label (or target output).

Since the LSMs were proposed for neuroscience applications, the nonlinear elements in the reservoir are based on spiking neurons. The spiking neurons are built on the idea that the information is transmitted once it reaches a specific value. When the neuron exceeds the threshed level, the neuron fires and generates a corresponding signal. One of the well-known spiking neuron models is the leaky integrate-and-fire model [58]. Thus, if the RC is constructed using an LSM structure with spiking neurons, it is called LSM-based RC; otherwise, it is referred to as an ESN-based RC.

2.2.1. Echo State Networks

Echo state networks have developed in the frame of machine learning applications. An ESN consists of an input layer with an input weight matrix of $\boldsymbol{W}_{in}$, which scales the temporal data and applies it to the fixed reservoir. Additional input can also be provided to bias the network, $\boldsymbol{W}_{bias}$. The reservoir is a random, fixed recurrent network of nonlinear nodes with a non-trainable, internal weight matrix of $\boldsymbol{W}_{res}$, driven by a temporal input, $\boldsymbol{u}(n)$;

where n = 1, 2, 3, . . . indicates the different data points in the dataset. The reservoir states, ***x**(n)*, are the nonlinear mappings of the input signals. Nonlinear expansion of the input to a higher dimension space is widely used in ML algorithms, including support vector machines (SVMs). This nonlinear expansion function that transforms the input into a high-dimensional vector is usually referred to as 'kernel'. Therefore, one may introduce RC as a nonlinear expansion approach, which uses the nonlinear dynamic reservoir as a kernel with a memory of the input history as

$$\boldsymbol{x}(t) = \boldsymbol{f}\left(\boldsymbol{W}_{in}\,\boldsymbol{u}(n) + \boldsymbol{W}_{\boldsymbol{res}}\boldsymbol{x}(n-1)\right) \tag{2-3}$$

where ***x**(n)* represents the vector of the reservoir states at the discrete timestep *n,* $\boldsymbol{x}(n-1)$ is the reservoir states' vector in the previous time step, ***f*** is the nonlinear activation function (typically the hyperbolic tangent, *tanh*, or another sigmoid function), $\boldsymbol{W}_{in}$ and $\boldsymbol{W}_{\boldsymbol{res}}$ are the matrices of input and internal network connections, respectively. The formula considers bias terms to reduce the written complexity implicitly. A more general model can include the possibility of feedback connections ($\boldsymbol{W}_{back}$) from the output layer units to the reservoir

$$\boldsymbol{x}(n) = \boldsymbol{f}\left(\boldsymbol{W}_{in}\,\boldsymbol{u}(n) + \boldsymbol{W}_{\boldsymbol{res}}\boldsymbol{x}(n-1) + \boldsymbol{W}_{back}\,\boldsymbol{y}(n-1)\right) \tag{2-4}$$

where ***y**(n)* is the weighted sum of the internal reservoir states, and ***y**(n - 1)* is the most recent output.

The trainable readout layer with the weight matrix $\boldsymbol{W}_{out}$ is trained and reconfigured for various temporal applications. The state matrix ***x*** is then utilized to get an estimation of the desired output function $\boldsymbol{y}_{target}$, as:

$$\boldsymbol{y}(n) = \boldsymbol{W}_{out}\,\boldsymbol{x}(n) \tag{2-5}$$

In some networks, a constant bias term is implemented in the ***x**(n)*, and W_{out} must contain a corresponding column of weights. On the other hand, some include the input ***u**(n)* as an extra feature in ***x**(n)*, corresponding to a direct connection from the input to the output layer. The goal is then to minimize the error between the weighted linear sum of reservoir states ***x**(n)* trained on the linear regression of reservoir states ***x**(n)* and target signals $\boldsymbol{y}_{target}(n)$. The readout layer is trained through any least-squares matrix solution

such as ridge regression, also called Tikhonov regularization with the regularization constant β used to penalize large $\boldsymbol{W}_{out}$, which is defined as:

$$\boldsymbol{W}_{out} = \boldsymbol{Y}_{target}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \beta\boldsymbol{I})^{-1} \quad (2\text{-}6)$$

2.2.2. Physical RC Models

In the physical domain, models are more complex compared to software-based models. The specific form of the reservoir model varies depending on the physical system being studied. Let us consider an example of photonic RCs [34]. In this case, the physical system under study involves the propagation of light through an optical medium, such as a waveguide. In photonic RC, the input signal is usually encoded as variations in the intensity or phase of the optical signal. The input signal can be modulated using different techniques, such as intensity modulation or phase modulation. In an intensity modulation scheme, for example, the input signal is used to control the power or intensity of the optical signal propagating through the waveguide. This can be achieved using various methods, including electro-optic modulation or optoelectronic feedback loops. The optical signal, after being modulated by the input signal, propagates through the waveguide and interacts with various elements, such as optical amplifiers, couplers, and nonlinear elements. These interactions introduce nonlinear transformations to the optical signal, making the waveguide act as a nonlinear reservoir. The output of the reservoir is then measured using photodetectors or other optical sensors. The measured output can be processed further to extract useful information or perform specific tasks such as pattern recognition, time-series prediction, or classification.

The physical reservoir model in photonic RC can be described using equations that capture the behavior of light propagation through the waveguide, taking into account the nonlinearities and interactions with the optical elements. These equations may include terms that describe the input modulation, the optical propagation, and the dynamics of the optical elements in the system [34] as

$$\boldsymbol{x}(n) = \boldsymbol{f}\big(\boldsymbol{W}_{in}\,\boldsymbol{U}(n) + \boldsymbol{W}_{\boldsymbol{res}}\boldsymbol{x}(n-1)\big) \quad (2\text{-}7)$$

where $\boldsymbol{U}(n)$ can be the modulated optical signal that carries the input information.

Another example is the origami-based RC. Origami-based RC is a physical computing platform that uses folded origami structures as the reservoir for processing input signals [52]. In this system, the goal is to transform an input signal, $u(n)$, which represents a desired pattern or behaviour, into mechanical movements of the origami structure, $x(n)$. This input signal is used to modulate or manipulate the behaviour of the origami structure. Input signals may involve applying external forces to the origami structure such as tension, compression, or pressure. However, in some structures, an actuation mechanism is implemented to deliver input, $u(n)$, to the reservoir [49]. In angle encoding techniques, mechanical actuators such as motors or servos can be integrated into the origami structure [52]. These actuators can manipulate specific parts of the origami to achieve desired angular changes in response to input signals. Therefore, the input signal, $u(n)$, undergoes mechanical transformation (denoted as $U(n)$) before driving the origami-based reservoirs. The configuration of the origami structure, then, changes over time due to mechanical folding and unfolding, and these dynamic transformations introduce nonlinear dynamics into the system. The output of the reservoir was measured using sensors that track the position or shape of the origami structure as it unfolds and moves. The measured output is processed further to control the robotic crawling motion or other desired tasks.

Another example is physical reservoirs constructed using mechanical oscillators. Mechanical oscillators are physical systems designed to generate periodic motion or vibrations. They produce oscillations, which are repetitive back-and-forth movements or vibrations, and these oscillations can occur in various forms, such as mechanical vibrations or waves. In this case, the input signal to the reservoir is the driving voltage signal, $\boldsymbol{u}(\boldsymbol{t})$, which creates a driving force $F_d \sim \boldsymbol{u^2}(t)$. This force then acts on the oscillators, resulting in a displacement that can be measured using strain sensors (resistors). The measurement of this displacement is used as part of the reservoir model [25]. Hence, the physical reservoir model can be described by the equation:

$$\boldsymbol{x}(n) = \boldsymbol{f}\left(\boldsymbol{W}_{in}\,\boldsymbol{u^2}(n) + \boldsymbol{W_{res}}\boldsymbol{x}(n-1)\right) \tag{2-8}$$

Comparing this equation with the previous equation (2-3) for software RC, we can observe that the input signal to the physical reservoir has a different representation in this context. The input signal, $\boldsymbol{u}(n)$, in the form of voltage signal gets squared to generate the

required force to drive the oscillator, multiplied by the input weight matrix and then combined with the weighted output from the previous time step before being passed through the activation function. This highlights the distinction between the software-based RC models, where the input signal is typically provided directly, and the physical reservoir models, where the input signal may undergo transformations based on the characteristics of the physical system being studied.

2.2.3. Performance Evaluation

The process of implementing an RC for temporal tasks includes a specific nonlinear transformation, $\boldsymbol{y}_{target}(n)$, of an input signal, $\boldsymbol{u}(n)$. In a temporal task $\boldsymbol{y}_{target}(n)$ and $\boldsymbol{u}(n)$ represent signals in a discrete-time domain, and the desired output function may have a memory of previous input values. In a temporal task, the function to be learned depends on the input history: $\boldsymbol{y}(n) = \boldsymbol{y}(\boldsymbol{u}(n), \boldsymbol{u}(n-1), \boldsymbol{u}(n-2), \ldots)$. Different functions can be used for measuring the error, such as the mean squared error (MSE), root mean squared error (RMSE), the normalized mean squared error (NMSE), or the normalized root mean squared error (NRMSE) [59]

$$MSE(\boldsymbol{y}, \boldsymbol{y}_{target}) = \langle \left\| \boldsymbol{y}(n) - \boldsymbol{y}_{target}(n) \right\|^2 \rangle \tag{2-9}$$

$$RMSE(\boldsymbol{y}, \boldsymbol{y}_{target}) = \sqrt{\langle \left\| \boldsymbol{y}(n) - \boldsymbol{y}_{target}(n) \right\|^2 \rangle} \tag{2-10}$$

$$NMSE(\boldsymbol{y}, \boldsymbol{y}_{target}) = \frac{\langle \left\| \boldsymbol{y}(n) - \boldsymbol{y}_{target}(n) \right\|^2 \rangle}{\langle \left\| \boldsymbol{y}(n) - \langle \boldsymbol{y}_{target}(n) \rangle \right\|^2 \rangle} \tag{2-11}$$

$$NRMSE(\boldsymbol{y}, \boldsymbol{y}_{target}) = \sqrt{\frac{\langle \left\| \boldsymbol{y}(n) - \boldsymbol{y}_{target}(n) \right\|^2 \rangle}{\langle \left\| \boldsymbol{y}(n) - \langle \boldsymbol{y}_{target}(n) \rangle \right\|^2 \rangle}} \tag{2-12}$$

where $\langle \ldots \rangle$ stands for the mean and $\| \ldots \|$ denotes the Euclidean distance.

2.2.4. On Nonlinearity and Memory

Memory refers to the neurons/nodes' capability to store information in a system. Fading memory, also known as short-term memory, is present in almost any physical

system. Fading memory is required to satisfy the so-called echo state property (ESP), which Jaeger introduced for ESNs: the system must forget its previous states as time passes [60]. That is, the current state is a pure injective function *E* on all previous inputs as

$$\boldsymbol{x}(n) = E(\ldots, \boldsymbol{u}(n-1), \boldsymbol{u}(n)) \tag{2-13}$$

meaning that the network acts as a fading memory where the states contain information from previous time steps. However, the most weight is given to recent inputs. In an RC system, we require fading memory terms long enough to store information from the recent past but short enough to forget the distant past. Usually, the memory is implemented digitally, while in physical RC, there are suitable alternatives that can be a reliable source of memory.

The nonlinearity of nodes inside a reservoir can be of different types. For example, spiking neuron models are utilized in RC based on liquid-state machines. However, analog, differentiable functions such as sigmoid and hyperbolic tangent (*tanh*) are primarily employed in RCs based on ESNs. Alternatively, some functions that are widely used in NNs, such as rectified linear unit (ReLU), leaky ReLU, and exponential linear unit (ELU) for ESN-based RCs, can be employed in these structures, as illustrated in Figure 2.2. The first two functions are the traditional nonlinear functions used in ML, while the last three are among the nonlinearities used in more recent ML. While some nonlinearities can be achieved in many saturable physical systems or devices, other ones can be obtained from diode-like circuits. Simply including nonlinear functions in a node's activation function does not adequately communicate the importance of input range in facilitating nonlinear behaviour. A nonlinear element may act linearly if the input range is too small or set to its linear region, as shown in Figure 2.3. A reservoir built using several nonlinear nodes reacting linearly to the input range is called a linear reservoir. The nonlinear reservoir is defined as a network in which most of its internal nodes have nonlinear responses with respect to a given input range. In a saturated reservoir, most of the nodes are driven into the saturation region for a given input range.

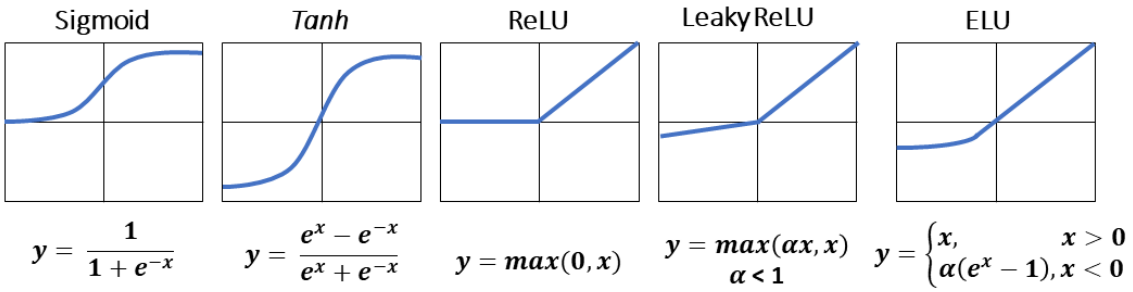


Figure 2.2. Various forms of nonlinear functions in a reservoir.

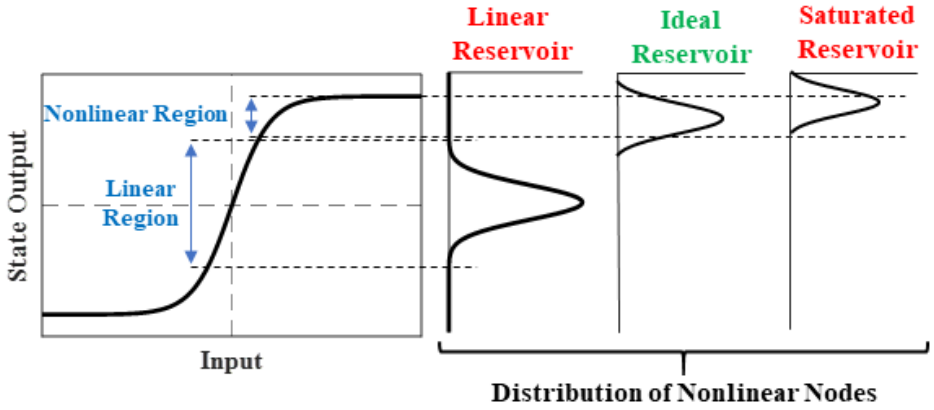


Figure 2.3. Examples of reservoirs in an RC system.

2.2.5. Global Parameters of RC

One should adhere to certain guidelines to create a high-quality reservoir in software-based reservoir computing. These include generating a vast reservoir comprising numerous nonlinear nodes (ranging from tens to thousands) that are sparsely and randomly connected (with connectivity typically set at 1-20%). It's important to note that multiple factors impact the performance of reservoir computing systems, such as:

Reservoir size is the number of nonlinear nodes in a reservoir that directly contributes to the weighted output sum of the RC. Larger reservoirs produce a higher dimensional state space, promising better performance.

Input scaling refers to the process of scaling the input weight matrix used to connect the input to the internal nodes of the reservoir. It is an important step in the design of an RC system as it directly affects the performance and behaviour of the reservoir. The input weight matrix is typically randomly generated, and values are drawn from a specific distribution. For uniformly distributed matrices, the input scaling factor, *a,* is defined as the

range of the interval [−*a, a*] from which values of the input matrix are sampled. The standard deviation is often used as a scaling measure for normally distributed input weights. The goal of input scaling in RC is to create a reservoir where most of its internal states perform nonlinearly for a given input. This is achieved by selecting an appropriate scaling factor that shifts the reservoir to operate in its linear, nonlinear, or saturation regions. If the scaling factor is too small, the input signal may not be strong enough to drive the reservoir to perform nonlinearly, resulting in poor performance. On the other hand, if the scaling factor is too large, the reservoir may become saturated, causing the performance to degrade. Therefore, the choice of the scaling factor is critical for achieving optimal performance in an RC system. The scaling factor can be chosen based on the specific application and the characteristics of the input data.

$$\boldsymbol{W}_{in}{}' = a\,\boldsymbol{W}_{in} \tag{2-14}$$

The spectral radius is a measure of the fixed weights inside the reservoir, representing the connections between different nonlinear nodes. The spectral radius refers to the largest absolute value of the eigenvalues of the reservoir connection matrix, W_{res}. It is denoted by the symbol ρ and has a direct impact on the performance of RC systems. When the spectral radius is close to 1, the highest eigenvalue of the reservoir weights, the RC system provides an extended memory of input, making it suitable for applications requiring more information from past events. On the other hand, when the spectral radius is 0, the RC system becomes a feedforward network. Since the memory of past inputs is lost in this case, the system is unsuitable for applications that require past events.

The leaking rate, α, is a parameter used in RC to control the rate at which the current state of the reservoir decays and is replaced by the new state. It is often used in RC systems that include leaky-integrator nodes [61] with some memory of past events. In these systems, the state update equation is expressed as

$$\boldsymbol{x}(n) = (1-\alpha)\boldsymbol{x}(n-1) + \alpha\boldsymbol{f}\big(\boldsymbol{W}_{in}\,\boldsymbol{u}(n) + \boldsymbol{W}_{\boldsymbol{res}}\boldsymbol{x}(n-1)\big) \tag{2-15}$$

where $\boldsymbol{x}(n)$ is the current state of the reservoir, $\boldsymbol{u}(n)$ is the current input, $\boldsymbol{f}$ is the node's nonlinear function, and $\boldsymbol{W}_{in}$ and $\boldsymbol{W}_{\boldsymbol{res}}$ are the input and reservoir weight matrices, respectively. In a software-based RC, the leaking rate can be adjusted easily, and it is

proposed to set it to match the speed of the input dynamics and/or the target to provide longer short-term memory of the information [54].

The number of connections from the input to a reservoir node and the number of recurrent connections of internal nodes determine the sparseness of a reservoir. It is recommended to make the reservoir connections sparse enough (≈20% recurrent connection) [43] for improved performance.

Selection of input scaling, spectral radius, and leaking rate are essential for good performance and are pretty task-specific in software RC. However, many of these parameters require a thorough understanding of the problem and developing a suitable physical RC accordingly by considering external restrictions in physical RCs.

2.2.6. Reservoir Configurations

Standard ESNs consist of many simple processing units called nodes, organized in an arbitrary and fixed network of connections in their reservoir layer. The random connectivity of the nodes allows the reservoir to have rich, dynamic internal representations that are applied to solve a variety of tasks, such as time series prediction, speech recognition, and image classification.

Since the reservoir plays a vital role in the performance of RCs, researchers have studied and developed different reservoir topologies [21]. A study conducted in 2011 compared the performance of several topologies, including the delay line reservoir (DLR), DLR with feedback connections (DLRB), simple cycle reservoir (SCR), and leaky reservoir (LR) with standard ESNs, as shown in Figure 2.4 [54].

The DLR configuration comprises nodes organized in a line, with only a feedforward connection in the reservoir with the same connection weight. The DLRB topology has the same structure as DLR except for the feedback connections between a reservoir node and the preceding one in the DLRB. In this topology, all the feedforward connections have a similar weight, and all the feedback connections are set to a fixed value. The SCR topology organizes nodes in a cycle where all connections have the same weight. The LR topology consists of a standard ESN and a layer with leaky integrator nodes. A leaky integrator node, a first-order low-pass filter, follows each node in a standard reservoir.

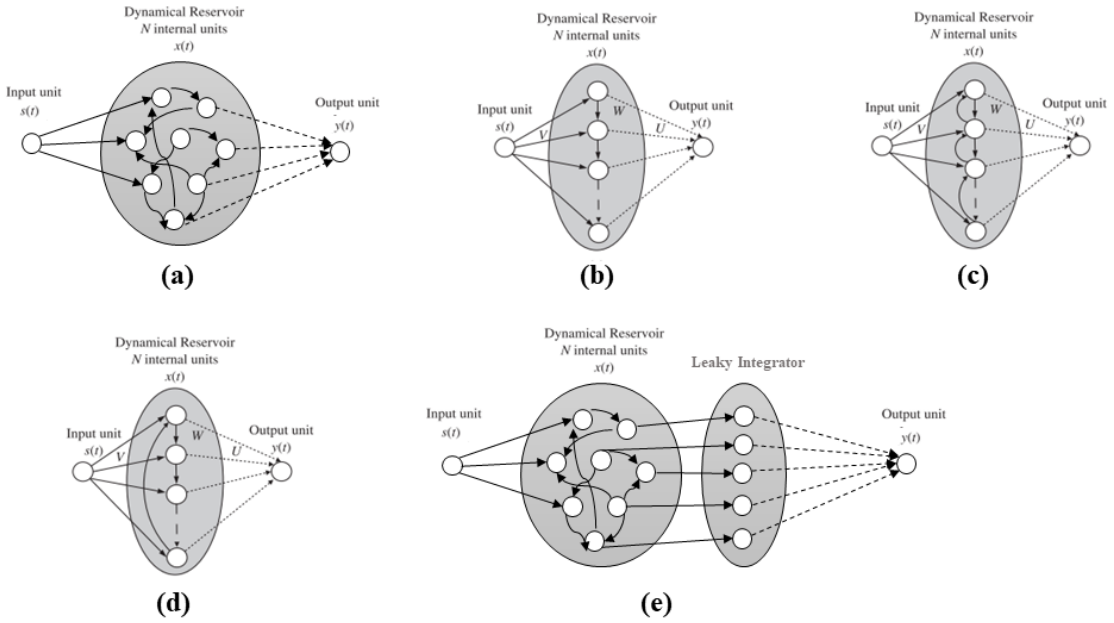


Figure 2.4. The representatives of (a) standard reservoir, (b) delay line reservoir, (c) delay line with feedback, (d) simple cyclic reservoir, and (e) leaky reservoir © 2011 IEEE [21]**.**

The study found that all the structures performed similarly on a wide range of time series benchmarks, suggesting that ordered and random connections offer similar performance. However, each topology has advantages and disadvantages, such as ease of implementation, computational efficiency, or ability to handle specific data types. Therefore, the choice of topology for a reservoir should be based on the specific requirements of the task at hand.

Another type of reservoir was developed to tackle some practical implications in physical RC, known as single dynamical node (SDN) RC. SDN RC uses a dynamical system with a single nonlinear node coupled to itself through a delay line. Also known as delay-based RC systems [62], the nonlinear nodes in this reservoir are multiplexed in time and must be retrieved sequentially in the time domain in the readout layer. The nonlinear nodes in an SDN reservoir are called virtual nodes, as unlike the spatially distributed nodes in classical reservoirs (ESNs and LSMs), these virtual nodes are temporally distributed. Delay-based RC is practically useful in the physical realizations of RC systems and has been utilized in different physical RC systems. Mathematically speaking, the SDN reservoir is integrated into the sampled solutions of a single delay differential equation (DDE) as [23]:

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{F}\big(t, \boldsymbol{x}(t), \boldsymbol{x}(t-\tau)\big) \tag{2-16}$$

where *t* represents continuous time, ***x*** is the vector of the reservoir states, ***F*** is a function determining the flow of this system, and $\tau > 0$ is the delay period. DDE fundamentally differs from ordinary differential equations as the time-dependent solution of a DDE is not uniquely determined by its state at a given moment. The general concept of this approach is depicted in Figure 2.5, which illustrates how the recurrent reservoir network is realized using the single nonlinear dynamical node (NL) with delayed feedback. The *N* virtual nodes in the reservoir are distributed uniformly in time over the delay interval τ, with each virtual node having a width of $\theta = \tau/N$. The physical nonlinearities of the nodes generate the reservoir states ***x****(t)* at the end of each time segment. These states are a combination of both the memory and the nonlinear transformation of the input signal, as they are integrated into the transient response of the nonlinear node to a specific input at a given time. To generate the random virtual nodes, a masking signal, *m(t)*, is combined with the discrete input signal, with each time segment having a width of θ. This approach simplifies the system structure but comes at the cost of slower processing speed compared to spatially distributed reservoir computing. To compensate for this disadvantage, the dynamical behavior in the system must operate at *N* times the higher rate than in a spatially distributed reservoir with N internal states.

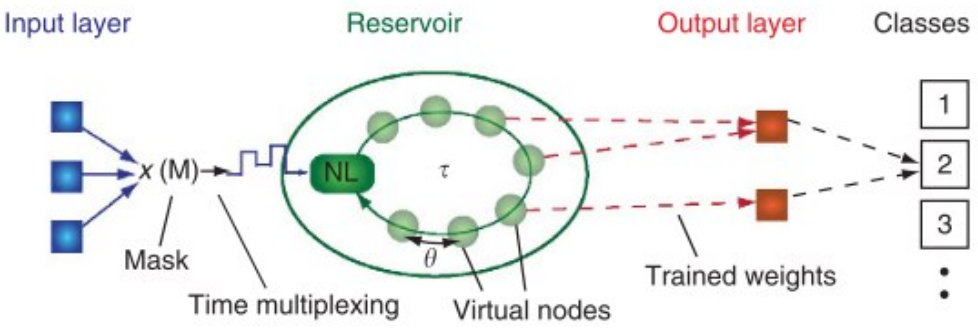


Figure 2.5. Schematic view of a reservoir computer based on a single nonlinear node (NL) with delay (τ). Virtual nodes are defined as temporal positions in the delay line, Copyright © 2011, L. Appeltant *et al.* [62]**.**

2.2.7. RC Evaluation Tests

Reservoir computing has been applied to and optimized for many applications, from classification to time series prediction. Some benchmark tasks have been developed

to assess the performance of RC for these applications [23], summarized in Table 2-1. The RC system's input and target output signals are determined based on the selected task.

In a **pattern classification** task, the input is a time series, while the discrete output value represents the class label. The input and output signals are temporal data in the time series prediction.

Temporal XOR Task

Temporal XOR is an XOR function which is integrated in time. The input is a random sequence of binary or real values, and the output is the XOR (or multiplication) of the input signals at a given time t, $\boldsymbol{u}(t)$, and the one before $\boldsymbol{u}(t-1)$: $\boldsymbol{y}(t) = \boldsymbol{u}(t) \cdot \boldsymbol{u}(t-1)$. Prediction of this temporal benchmark requires a network's nonlinearity and memory properties. Training the network makes it possible to perform more delayed operations, which are useful for applications that require a larger memory of past information. In such cases, the network needs to remember inputs from further back in time. The XOR operation can be computed for the inputs $\boldsymbol{u}(t-k)$ and $\boldsymbol{u}(t-k-1)$, where k is an integer representing the delay size. Importantly, the reservoir dynamics are independent of the RC system's output.

Table 2-1. Applications and related evaluation tests on RC

Problems	Benchmark tasks
Pattern classification	Spoken digit recognition [63] Waveform classification [64] Human action recognition [65] Handwritten digit image recognition [66]
Time series prediction	Chaotic time series prediction [56] NARMA time series prediction [67]
Pattern generation	Wave generation [68] Limit cycle generation [69]
Adaptive filtering and control	Channel equalization [70]
System approximation	Temporal XOR task [71] Temporal parity task [71]
Short-term memory	Memory capacity [60]

NARMA Task

The performance of RC systems can be examined using the nonlinear autoregressive moving average (NARMA) task. NARMA is a discrete-time temporal task with an n^{th}-order time lag. The NARMA time series is given by [72]

$$\boldsymbol{y}(t) = \alpha\, \boldsymbol{y}(t-1) + \beta\, \boldsymbol{y}(t-1) \sum_{i=1}^{n} \boldsymbol{y}(t-i) + \gamma\, \boldsymbol{u}(t-n)\, \boldsymbol{u}(t-1) + \delta \qquad (2\text{-}17)$$

where α = 0.3, β = 0.05, γ = 1.5, and δ = 0.1 [72]. If we consider the NARMA task as a dynamical system whose output is given as equation (14), the stability of the system, its divergence to infinity, and its capability to store input streams, are determined by the parameters above. Thus, these values are constant for this task to keep the NARMA in a determined dynamical region. The input ***u(t)*** is a uniform distribution in the interval [0,0.5]. ***u****(t − 1)* is a one-step delayed input, ***y****(t)* is the current approximation of the system (using linear readout), and ***y****(t − 1)* is the prediction at the output of the RC system once ***u****(t − 1) is given as input*. The dependence of NARMA on its nonlinearity and long-time lags makes it a challenging problem for any computational system. For example, calculating NARMA10 requires a device capable of algorithmic programming and perfect memory of the input and the outputs of up to 10 previous time steps.

Waveform Classification

The waveform classification problem involves identifying different types of waveforms, such as sinusoidal, triangular, and square waves. It is a fundamental task in classification. Multiple waveform data with different frequencies are first generated to tackle this problem. The data can be presented to the RC system as an electrical signal or physical stimulation. Each waveform type is assigned a class label, and the linear readout layer is optimized to recognize the patterns in the data and correctly classify the waveforms.

Memory Capacity

The short-term memory capacity of the RC system is essential in applications requiring memory of past events. The concept of memory is based on the RC system's ability to retrieve past information from the reservoir using the linear combinations of its internal states. To evaluate the short-term memory capacity, we compute the *k*-delay memory capacity (MC_k) may be computed, as introduced and derived in [60]

$$MC_k = \frac{cov^2\big(\boldsymbol{u}(t-k), \boldsymbol{y}_k(t)\big)}{\sigma^2\big(\boldsymbol{u}(t)\big)\ \sigma^2\big(\boldsymbol{y}_k(t)\big)} \tag{2-18}$$

where $\boldsymbol{u}(t-k)$ is a k-step delayed input and $\boldsymbol{y}_k(t) = \boldsymbol{u}(t-k)$ is its reconstruction at the output of the RC system. *cov* represents the covariance of the two under-study time series, and σ^2 is the variance of the time series signal, either the input or the output of the linear readout layer. The overall short-term memory capacity is then approximated as:

$$MC = \sum_{k=1}^{k_{max}} MC_k \tag{2-19}$$

2.3. Sample Physical RC Platforms

2.3.1. Photonic RC

The history behind photonic RC goes back to 2008 when a research group implemented a photonic RC using an on-chip network of semiconductor optical amplifiers (SOAs) distributed in a cascade way [73]. The power-saturation behaviour of an SOA is naturally a nonlinear function (*tanh*). As the first hardware realization of a photonic RC, the same group developed a passive, linear photonic reservoir using optical waveguides, optical splitters, and combiners on a silicon-on-insulator substrate (SOI), serving as a complex interferometer [74]. The existing technology fails at injecting inputs and making measurements at the rate of Gbit/s. Thus, the photonic reservoirs mostly used a photonic delay line (such as a spiral waveguide) to be integrated with the nodes to achieve slower dynamics. On the other hand, the system requires a nonlinearity to highly expand the input states, which was realized using a fast photodetector that detects the optical power, resulting in the reservoir states as $\mathrm{X_n} \propto \|\mathrm{E}^2\|$, where E is the optical field. Incorporating photodetectors in optical systems introduces an element of non-pure optical hardware, as the conversion of optical signals to electrical signals occurs within the detector.

The first optical hardware implementation of a delay-based RC was based on an optical ring cavity [64], as shown in Figure 2.6. This optoelectronic RC uses a Mach-Zehnder modulator for the nonlinear modulation of the incoming light ($\sin^2$), the system dynamics was employed as the source of the fading memory, and a fibre spool is used to provide the feedback delay line. In an all-optical delay-based RC, the nature of the input

and the reservoir is optical. The first two experimental implementations of this type were based on active devices. In one, SOA (providing the *tanh* nonlinearity) was placed in an optical ring cavity [75]. In another, a semiconductor laser (providing the ReLU-like nonlinearity) was located in a loop with a feedback delay line [37]. However, passive devices are an essential step towards developing high-speed, low-consumption, photonic computers because they do not require a power supply, making them energy-efficient and ideal for high-speed operation.

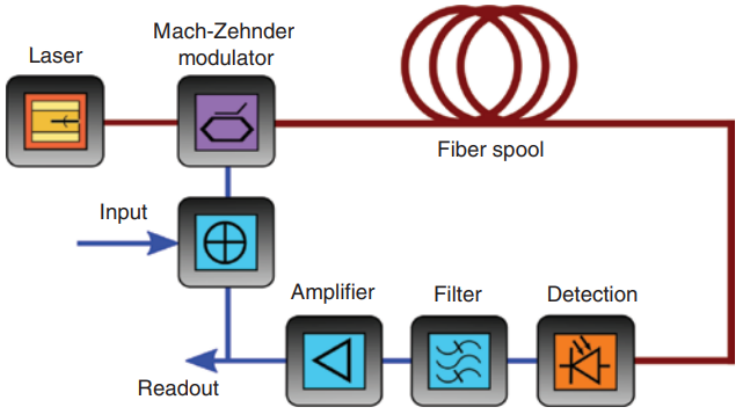


Figure 2.6. Scheme of the optoelectronic reservoir computer, Copyright © 2012, Y. Paquot *et al.* [64]. The optical (electronic) path is depicted in red (blue) colour.

A delay-based photonic RC system based on a semiconductor saturable absorber mirror placed in a ring-like optical cavity has been demonstrated [76], in which the external input modulates a light-emitting diode (LED), providing the injected light. Alternatively, a simple linear fibre cavity can be used as a reservoir computer if the output layer is nonlinear, such as detecting the propagated optical field inside the cavity using a photodetector. Apart from the photonic reservoirs, some researchers were focused on the physical implementations of the entire computers using photoelectronic devices [34], i.e., the input layer, readout layer and the reservoir itself. Summarized information on nonlinear and memory elements and the coupling mechanism is provided in Table 2-2.

2.3.2. Memristors and Atomic Switches

The three fundamental electrical components (resistors, capacitors and inductors) create relationships between current, voltage, charge, and magnetic flux. Memristors link

the charge and flux and were first theorized in 1971 by Chua [77]. The first nano-scale memristors were demonstrated in 2008 [77]. Two equations define memristors

$$V = R(x, I).I \tag{2-20}$$

$$\frac{dx}{dt} = f(x, I) \tag{2-21}$$

where *V* and *I* are the voltage and currents of the device, *R* is the resistance, the function *f* is device-specific, and the variable *x* is a state variable used as a mathematical analogy of the physical changes within the device.

Table 2-2. Nonlinearity, memory, and coupling in photonic RC

Optical Element	**Input-Output Signal**	**Nonlinearity Type**
Mach Zehnder modulator [40], [64]	Optical - Optical	Sin^2
Semiconductor optical amplifier [73]	Optical - Optical	Tanh
Semiconductor laser [37]	Optical – Optical	Diode nonlinearity
Photodetector [33]	Optical - Electrical	Quadratic
Optical limiter [78]	Optical – Optical	Sigmoid
Fading Memory		
Fibre optics spool [64], Laser's relaxation oscillations in SL [37], Fibre cavity, Multimode ring resonator [78]		
Coupling in spatially distributed RC		Coupling in temporally distributed RC
Multimode interferometers (MMIs) [33], laser-to-laser coupling using spatial light modulator (SLM) [79]		Multiplexed in time [64]

Atomic switches [80] switch between high- and low-conductance states with a negligible intermediate transition. Atomic switches have a current threshold at which the switch breaks. Due to the random fabrication of atomic switch networks, they are easier to produce than memristor networks.

There is a wide range of research to exploit computing capabilities from memristors and atomic switches. In one approach, researchers created a memristive reservoir of silver nanowires [81]. This network contains different patterns within the various sections of the network, which can be combined using a readout layer for computations. Another research explored the possibility of reservoir computing using silver nanowire atomic switches [11]. Their networks have been used to generate higher harmonics of the input waves and generate various waveforms of the same frequency, highlighting the advantages of

network dynamics. The studies were then directed towards the topologies of memristor networks, the design of memristors operating in both the “learning” and “predicting” modes instead of continuous conductance updates, and the effectiveness of arbitrary memristor networks.

2.3.3. Mechanical RC

A mechanical model available as a physical reservoir is a network of mass-spring systems, which can be regarded as coupled mechanical oscillators. A mass-spring network reservoir where mass points are randomly connected to neighbouring mass points via nonlinear springs was first proposed in [24]. When each oscillator is described with a first-order ordinary differential equation (ODE), a system of *N* coupled oscillators can be described in the following general form

$$\frac{dx_i(t)}{dt} = F\big(x_i(t)\big) + G\big(x_1(t), \dots, x_N(t)\big), \quad for\ i = 1, \dots, N \tag{2-22}$$

where *t* represents continuous time, $x_i(t)$ is the state of oscillator *i* at time *t*, *F* determines the dynamics of isolated oscillators, and *G* is a coupling function. The input signal is given to some randomly chosen nodes as the external force, inducing nonlinear responses of the mass-spring oscillators. The output signal is obtained from a linear combination of the lengths of the springs, for instance. Simulations demonstrated the computing power of RC based on the mass-spring network in time series approximation [24].

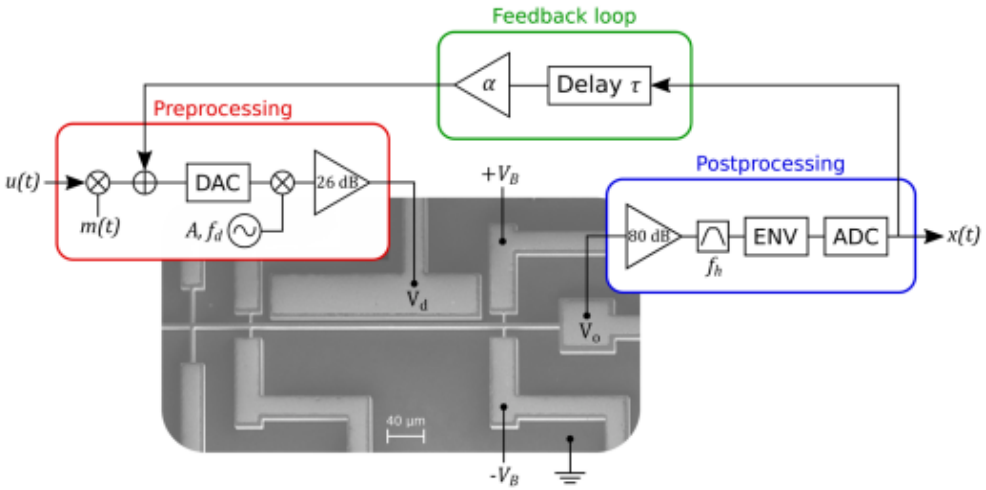


Figure 2.7. The proposed RC system based on a single beam and SEM image of the device [25]**.**

Micro-electro-mechanical systems (MEMS) technology can reliably produce small and energy-efficient devices exhibiting rich, dynamic behaviour, promising alternatives to computing using conventional electronics [82]. MEMS resonators often demonstrate nonlinear behaviour when driven above a certain critical amplitude. When oscillating at large amplitudes, the structure's stiffness changes result in nonlinear resonance frequency shifts. Frequently, the Duffing equation for nonlinear oscillators is used to describe the motion of MEMS resonators.

In their work, Dion et al. developed reservoir computers using a network of virtual nodes multiplexed in time, as depicted in Figure 2.7 [25]. The network utilizes a single oscillating silicon beam, which exhibits a classical Duffing nonlinearity, serving as the system's source of nonlinearity. This innovative approach enables using a single physical component to generate the nonlinearity required for reservoir computing, simplifying the system and reducing its complexity. The reservoir is electrically stimulated using amplitude modulation. The input signal (amplitude A and frequency f_d) is preprocessed in the digital domain, modulated, and amplified before being supplied to the drive electrode. Measuring the response involves amplifying the piezoresistive signal, bandpass filtering it around f_h, detecting its envelope, digitizing it, and reinjecting it with a delay and a gain in the preprocessing stage. The bandpass filter is locked at $f_h = 4f_d$ due to the quadratic relation between the applied force to the beam, the drive signal, and the frequency shift in the stiffed beam.

Later, a delay-based RC was developed to process the acceleration information provided by an inertial mass [12]. The device consists of an inertial mass electrostatically coupled to an oscillating beam through a gap. The motion of the inertial mass modulates an AC electrostatic field that drives the beam in its nonlinear regime. This nonlinearity is then used to implement machine learning in the mechanical domain.

2.3.4. In-materio Computing

In 2012, the European discovery project 'NASCENCE' was established to explore the computational properties of materials. NASCENCE employed a hybrid approach to create reservoir computers, combining computer and physical domains [41], [48], [83]. Reservoirs were configured using physical substrates consisting of single-walled carbon nanotubes (SWCNTs)/polymer composites spin-coated onto a PCB electrode array, as

illustrated in Figure 2.8 (a). Different materials were investigated, including SWCNT/PBMA, SWCNT/LC (liquid crystal), and gold nanoparticles, with the reservoirs eventually being based on SWCNT/PBMA composites.

The behaviour of the SWCNT/PBMA composite was found to be nonlinear for low concentrations of SWCNTs, with the nonlinearity related to the percolation of conducting pathways within the composite [84]. The network's time scale or dynamics were determined by how fast the SWCNTs could create the paths due to the applied electric field. The coupling between nodes (i.e., specific electrodes) was determined based on random conductive pathways. An evolutionary algorithm in the computer domain was used to determine the optimal configuration of the substrate, the control voltage signals, and the locations of the input/output electrodes. The experimental results showed that these substrates could be configured and trained as reservoir computers.

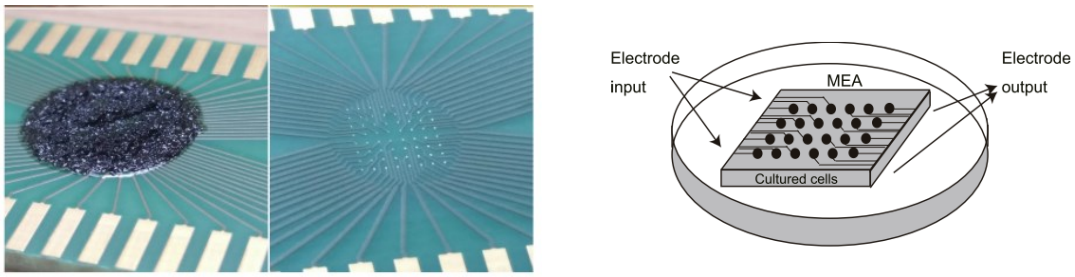


Figure 2.8. (a) Hardware reservoir system and CNT/polymer deposited onto PCB electrode array, © 2019 M. Dale *et. al.* [84]**. (b) a reservoir of in-vitro cell cultures, Copyright © 2007 Elsevier B.V. All rights reserved.** [85]**.**

2.3.5. Biological Computers

Some researchers developed *in vitro* RC with biological nodes for biological information processing. Microelectrode arrays (MEAs) are the main substrates to stimulate cultured cells or bacteria and measure their responses electrically [85]. A sample reservoir of *in vitro* cell cultures is shown in Figure 2.8 (b). In 2015, Obien et al. developed an RC based on cultured cells on MEAs, which was stimulated by two rectangular voltage pulses applied on the electrodes to generate action potentials of the cultured cells [85]. A leaky integrator was used to transform the MEA measurements into a time-continuous reservoir state, and a readout layer successfully classified the spike patterns. Some researchers have changed the direction toward the electrical stimulation protocols, where the influence

of low and high-frequency stimulations on living cortical networks was studied [86]. Some introduced optogenetic stimulation instead of electrical stimuli to control neural activities precisely [87]. Rat cortical neurons were cultured on MEAs as a reservoir in this study, and the reservoir was stimulated using random dot patterns. Finally, reservoirs based on other living organisms, such as the bacterium Escherichia coli (E. coli), were proposed [88], which react to different chemical inputs and generate complex temporal patterns.

2.3.6. Quantum Reservoirs

Reported quantum computers are based on qubits (or quantum bits), which contain information and can simultaneously be in the ground and excited states. The qubits considered for quantum reservoirs must represent a scalable physical system and be able to initialize their state. The scalability enables the building of larger reservoirs that can handle more complex computational tasks. If the qubits used are not scalable, building larger reservoirs with more qubits would be challenging, limiting the system's computational power. Additionally, the qubits should offer a coherence time longer than the gate operation time and have the capacity to conduct qubit-specific measurements [89].

Over the years, researchers have developed several physical realizations of qubits such as silicon qubits, nuclear magnetic resonance, ion traps, superconducting qubits, and Nitrogen vacancies in the diamond. Obst et al. introduced a nano-scale reservoir with quantum dots and chemical compounds that change their absorption spectrum depending on their environment's pH or redox potential [90]. An input signal is given as a change in the chemical properties of the compounds, which affect the signal transfer between quantum dots randomly dispersed in space, encoded as an emission pattern. Simulations confirmed the potential computational performance in an image recognition task.

2.4. Summary

The main advantage of physical reservoir computing is that it creates highly efficient and scalable computing platforms that can perform complex computational tasks in real-time. Additionally, physical reservoir computing can be implemented using various materials and technologies, making it a versatile and adaptable approach to machine learning. Physical RC is promising in bringing low-power data processing closer to the

sensors or even inside them. Whereas with RNNs deployed on hardware processors, the amount of power being consumed for loading the weight values or training the network especially when the model size is large is comparably higher than physical RC. However, there are also some limitations to physical reservoir computing, such as the need for precise control over the physical system, the potential for noise and other disturbances to affect the system's performance, and the difficulty of integrating physical reservoir computing with traditional computing architectures.

Chapter 3.

The Electro-Thermal Computing Platform

The development of physical computing platforms starts with the design of physical reservoirs. The design of physical reservoirs begins with exploring devices and materials that exhibit nonlinear responses to various types of input stimulation, such as physical, electrical, optical, or chemical. This initial step involves identifying materials that possess the intrinsic memory property, which refers to the time it takes for the material to respond to a specific stimulus applied for a short time and return to its initial state. Once the target material or devices has been identified, the next step is to implement a neuron that can operate with the appropriate input signal (*e.g.*, physical, chemical, electrical, optical, etc.) and produce target readings (*e.g.*, physical, chemical, electrical, optical, etc.). This process involves selecting the correct type of neuron and configuring it nonlinearly to respond to the specific stimuli that will be applied. Finally, the role of connections inside a reservoir is to enable the flow of information among the neurons. Information flows within the network through energy exchange among the neurons in the physical domain. This energy can take various forms, such as electrical or optical energy, and can be either already present in the material or explicitly created to flow in the reservoir. Finding the right balance of the energy that flows among neurons is crucial to get the most efficient results. This chapter partly contains my article published in *Advanced Materials Technologies* entitled *"*A Neuromorphic Electrothermal Processor for Near-Sensor Computing*"* [91]. All the steps taken to build a physical RC are explained in detail and the resulting physical RC is evaluated using standard tasks.

3.1. Development of a Physical Computing Platform

In the early stages of this research, we have been seeking new materials and devices for physical reservoir computing by exploiting the physics of sensors and sensor systems to go beyond linear readings of information and simplify the generation of context. One promising device for this purpose is the negative temperature coefficient (NTC) thermistor, a temperature-sensitive resistor. Thermistors are inexpensive devices used for temperature measurement and control in various applications. Their operating principle relies on the dependence of material resistivity on temperature. NTC thermistors are

generally made of nickel, cobalt, iron and silicon oxides, employed as pure elements or as ceramics and polymers. Thermistors such as bead, disk, and chip, as well as glass-encapsulated thermistors, are categorized according to their manufacturing processes. Disk NTC thermistors (used in this work) have metalized surface contacts and are featured as accurate, stable, and highly sensitive. NTC thermistors have several attractive properties for physical reservoir computing, including a wide resistance range, fast thermal response time, and high precision and accuracy. In this Chapter, we will investigate the use of NTC thermistors for physical reservoir computing and explore their potential for real-time data processing under standard tasks.

Nonlinearity

NTCs experience a reduction in their resistance when their temperature is increased. The resistance of an NTC thermistor as a function of temperature, T, is given by [92]

$$R(T) = R(T_0)\, e^{\beta\left(\frac{1}{T}-\frac{1}{T_0}\right)} \tag{3-1}$$

where T_0 is the nominal operating temperature, and β is a material constant. Current passing through a thermistor generates heat, raising its temperature above the environment. Its electrical power is calculated as $P_E = V.I$, where *I* and *V* are the is the current passing through and the voltage drop across the thermistor, respectively. Newton's law of cooling describes the heat transfer rate as [92]

$$P_T = K(T - T_0) \tag{3-2}$$

where *K* is the dissipation constant in mW/°C, which is a measure of the thermal connection of the thermistor to its surroundings. At equilibrium, these two rates (i.e., electrical and thermal power) must be equal, giving the current-voltage (IV) characteristic of the thermistor as:

$$V.I = -\frac{T_a\, K \ln\left(\mathrm{R(T)}/\mathrm{R}(T_0)\right)}{\left(\ln\left(\mathrm{R(T)}/\mathrm{R}(T_0)\right) + \beta/T_0\right)} \tag{3-3}$$

The IV characteristic of NTC thermistors exhibits interesting nonlinearities as a result of the self-heating of the device under large currents (see Figure 3.10 (a)). This

nonlinearity is a fundamental criterion in devices performing as reservoir nodes. When used as a sensor, the thermistor's resistance is measured using a small current to avoid self-heating. Under large currents, however, the device self-heats (Joule heating), which causes the resistance to decrease. If driven by a current source, the resistance drop continues until electrothermal equilibrium with the environment and the device reaches a stable operating point. When driven by a voltage source, the current through a device can continue to increase as its resistance drops due to self-heating. This can lead to potential damage to the device if the current is not limited through another mechanism.

Fading Memory

In addition to their nonlinear response to slow-changing signals, thermistors exhibit an interesting dynamic response to their input electrical excitation. This time-dependent response results from the heat-up and cool-down times of the thermistors, which are governed by the heat conduction dynamics of the medium (see Figure 3.1 (b)). This dynamic response provides the fading memory function.

When the thermistors operate in their self-heating mode, they show a dynamic response. The voltage across the NTC in response to an abrupt change in its current can be represented by

$$V_{LH}(\mathrm{t}) = V_{LH1}\left[1 - \exp\left(\frac{-t}{\tau_1}\right)\right] + V_{LH2}\ \exp\left(\frac{-t}{\tau_2}\right) \tag{3-4}$$

where t in this context shows a continuous time, and V_{LH1} and τ_1 represent the fast dynamics due to the abrupt increase in the current passed through the NTC. As a result of this change, the temperature of the NTC increases. Once the thermistor self-heats, its resistance decreases with a thermal time constant of τ_2. V_{LH2} expresses the change in the voltage until it reaches the equilibrium state. While the voltage for a decrease in the current will be:

$$V_{HL}(\mathrm{t}) = V_{HL1}\exp\left(\frac{-t}{\tau_1}\right) + V_{HL2}\ \left[1 - \exp\left(\frac{-t}{\tau_2}\right)\right] \tag{3-5}$$

Figure 3.1 (b) illustrates the dynamics of the NTC voltage due to the change in the current passing through it. This dynamic response can be utilized to leverage the desired fading memory.

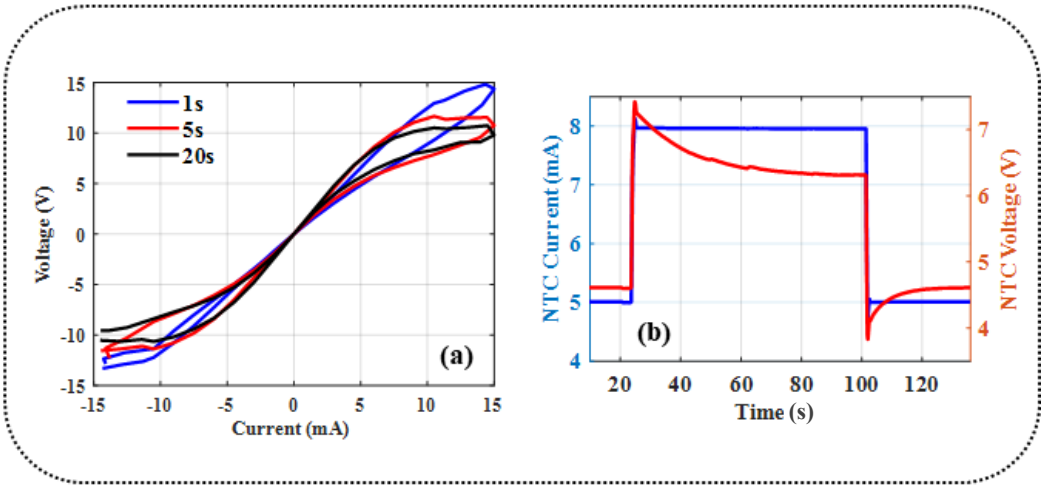


Figure 3.1. (a) V-I characteristics of an NTC thermistor (NT0515291) and (b) thermal dynamics of the NTC thermistor for changes in the current injected into the NTC.

The voltage across the NTC thermistor in response to an abrupt change in its current can be represented by a thermal time constant (TTC), a measure of the slow dynamics of the thermistor. The TTC is determined by the time it takes for the thermistor to reach ~63% of its maximum elevated temperature from the ambient temperature when it is driven into its self-heating mode. As highlighted earlier, τ_2 describes the thermal time constant of the NTC thermistor. The TTC for the thermistor (NT0515291) is reported in the datasheet and measured to be approximately 20 seconds.

Coupling

Coupling between thermistors requires a flow of energy between them which can happen in electrical or thermal domains. The first type of coupling is thermal coupling, when the heat generated by one thermistor can affect another nearby one. This can create recurrent nodes within the reservoir but is more effective on a micro scale when the thermistors are integrated onto a single substrate. The second type of coupling is electrical coupling, which can be achieved, for instance, by using a diode and a series resistance. Different types of electrical coupling structures can be used in different reservoirs.

Electro-Thermal Neurons

We developed active and passive neurons for potential use in a physical computing system. An active neuron uses nonlinear feedback in an amplifier with a

significant gain to produce a nonlinear output. As illustrated in Figure 3.2 (a), this topology consists of a high-gain amplifier and a nonlinear function in the feedback path.

$$Y = A\,(X - f(Y)) \tag{3-6}$$

where *X* and *Y* are the input and output of the system, respectively, *A* is the amplifier gain, and *f* is the nonlinear feedback. Since the feedback is nonlinear, the output will behave nonlinearly by considering a high gain, *A*, for the amplifier:

$$Y = f^{-1}(X) \tag{3-7}$$

Inverting/non-inverting amplifiers with an NTC thermistor in the feedback are potential options for such a system, as illustrated in Figure 3.2 (b). This way, the current flowing through the thermistor will be independent of the temperature and/or NTC thermistor. The system will directly transfer the nonlinearity of the thermistor to the output in magnitude. The coupling in a reservoir based on these nodes can be realized using a coupling resistor R_c. The coupling resistor can be adjusted to inject an attenuated version of the second physical state, x_2.

$$I_1 = \frac{\mathrm{u(t)}}{R_{in}}, \qquad I_c = \frac{x_2(t)}{R_c} \tag{3-8}$$

$$x_1(t) = -R_{NTC}(I_1 + I_c)\left(\frac{\mathrm{u(t)}}{R_{in}} + \frac{x_2(t)}{R_c}\right) \tag{3-9}$$

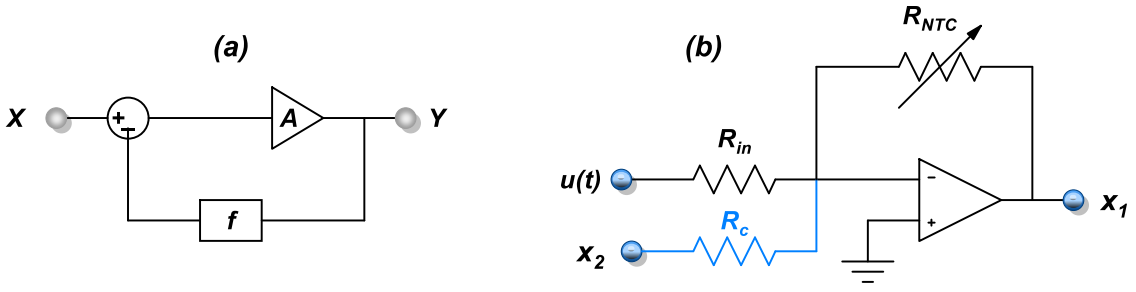


Figure 3.2. (a) A closed-loop system with dynamic, nonlinear feedback and (b) an inverting amplifier based on an NTC thermistor as the dynamic, nonlinear feedback. The coupling is shown in blue colour. Here, x_1 and x_2 are the output of neurons 1 and 2, respectively.

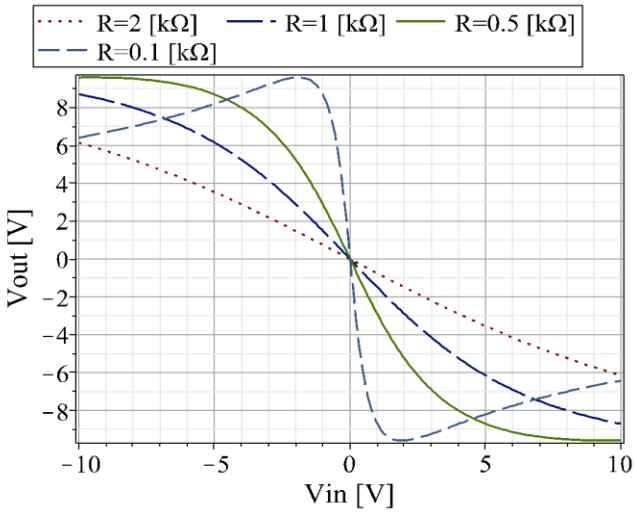


Figure 3.3. Characteristics of feedback-based nodes for various input resistance values.

As can be seen in Figure 3.3 (a), by changing the input resistance value, R_{in}, the current injected into the system is changed. As a result, the nonlinear region seems to be shifted, promising adjustable nonlinear nodes.

We have implemented an XOR gate using a reservoir composed of three active nonlinear nodes based on inverting amplifiers. The measurement results have demonstrated promising outcomes for the temporal XOR task. This achievement demonstrated the potential of reservoir computing for solving computationally challenging tasks using relatively simple hardware. The reservoir consisting of a network of nonlinear nodes can generate complex dynamics that enable efficient computation of a given task. In this case, implementing an XOR gate using a simple reservoir of only three active nonlinear nodes highlights the efficiency and effectiveness of reservoir computing.

The second active structure may be constructed by positioning the NTC thermistor in the drain terminal of a metal oxide semiconductor field-effect transistor (MOSFET), as shown in Figure 3.4 (a). By doing so, the MOSFETs nonlinearly convert the input voltage signal into a current signal. The nonlinear current combined with the thermal memory and nonlinearity from the self-heating of the NTC thermistors results in a highly nonlinear neuron. This approach offers several advantages, including increased computational power and improved accuracy. The use of a highly nonlinear neuron can enable more complex computations to be performed, leading to improved accuracy and better performance. The coupling, though, can be realized using another transistor, which injects

additional current into the NTC depending on the neighbour state's value.

$$I_1 = K_1(u(t) - V_{th})^2, \qquad I_c = K_c(x_2(t) - V_{th})^2 \tag{3-10}$$

$$x_1(t) = V_{DD} - R_{NTC}(I_1 + I_c)(K_1(u(t) - V_{th})^2 + K_c(x_2(t) - V_{th})^2) \tag{3-11}$$

where *K1* and K_c are the parameters specific to the transistor and can be scaled to shift the nonlinearity of nodes, V_{DD} is the supply voltage, and V_{th} is the threshold voltage of the transistor. Figure 3.4 (b) illustrates the nonlinear responses of the various nodes using various transistor sizes, K_1, with respect to the input.

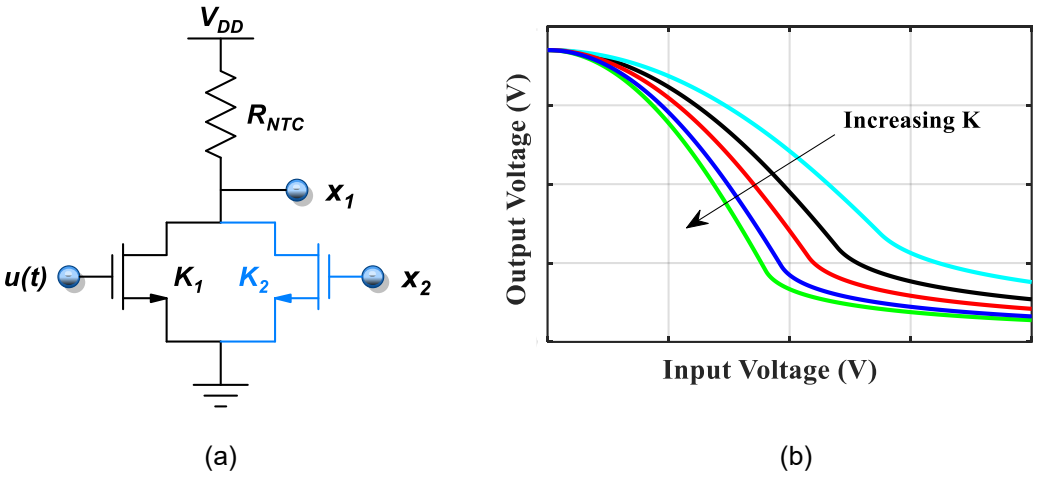


Figure 3.4. (a) An active node based on MOSFETs with nonlinear coupling and (b) characteristics of an active nonlinear node with varying width sizes for the MOSFET, K1. The coupling is shown in blue.

On the other hand, passive neurons do not have a gain or any other type of amplification and are driven by the input signal. They can be used to introduce nonlinearity into a system and offer simple and low-cost microfabrication/implementation. The simplest passive structure for reading the electrical signal at states is the resistive divider in a single branch, as shown in Figure 3.5 (a), where *x* is the state voltage. The neurons can be driven into various operating regions by adjusting the R_1. The possible coupling between the nodes can be realized using a diode and a resistor, as shown in Figure 3.5 (b), affecting the nonlinear response of both nodes. However, one may use coupling to create a delayed response. In that case, we may not be able to develop delayed states using passive structures, and we mainly use them to create various orders of nonlinearity.

The First Proposed Reservoir

It is important to consider specific factors and considerations to derive an analytical model for a reservoir in the proposed reservoir computing platform. The first factor is that the energy flow is one-directional due to the implementation of coupling using diodes. In a physical RC, diodes can be used to ensure that energy flows in a specific direction rather than being able to flow in both directions. The second factor considers that the electrical energy flows from a neuron with high energy to one with lower energy, similar to the concept of entropy in thermodynamics. This energy flow helps the network converge to a stable state. Energy exchange in such a physical RC depends on the input signal at each neuron (the amount of thermal energy generated by the neurons) and the diodes' direction. Thus, the coupling weights and the weights inside the reservoir are typically nonlinear, affecting how energy flows through the network.

When deriving an analytical model for the reservoir, it is essential to consider that even though the same device is used, its temperature and resistance may vary based on the input signal received at each neuron in the network. Additionally, the behaviour of the neurons may differ based on the bias resistors used. These factors must be considered when developing an accurate analytical model of the reservoir. However, the memory is constant for all the neurons inside the reservoir, so we may not be able to integrate various orders of memory into the network at this point. This can limit the network's ability to process complex input patterns or sequences if other solutions to this challenge are not introduced.

While a single neuron, as shown in Figure 3.6 (a), is formulated as:

$$V_{in} = V_j \left(1 + \frac{R_{NTC}(T_j)}{R_j}\right) \tag{3-12}$$

If the neuron (neuron *j*) drives another neuron (neuron *j+1*) with one branch of one-directional energy transfer, as shown in Figure 3.6 (b), the behaviour of the neuron *j* is described as:

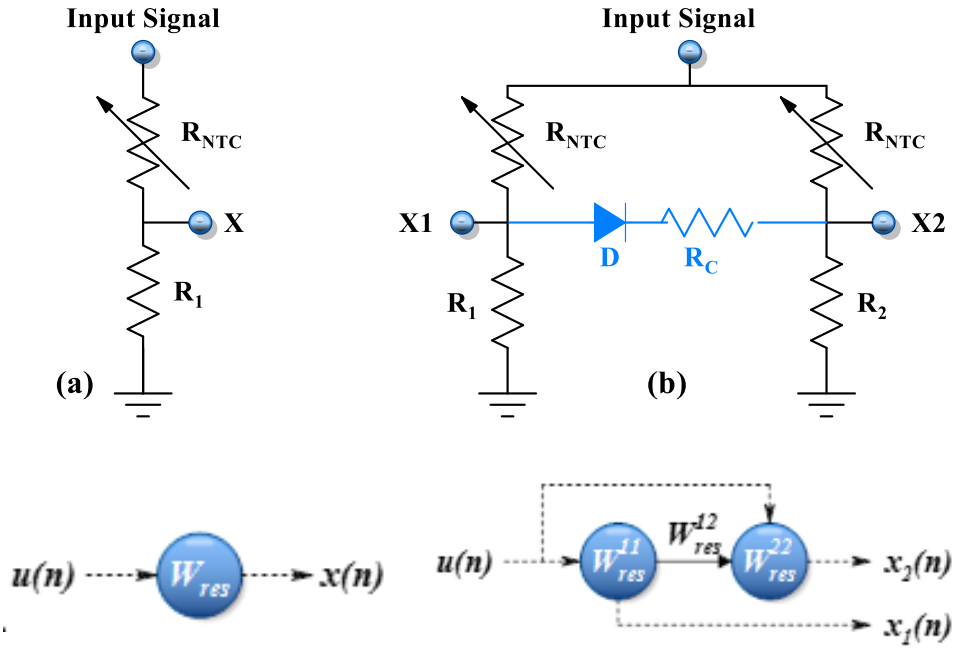


Figure 3.5. (a) A single branch NTC-based node, and (b) an example of coupled nodes.

$$V_{in} = \begin{cases} V_j\left(1 + \dfrac{R_{NTC}(T_j)}{R_j \| R_c}\right) - \dfrac{R_{NTC}(T_j)}{R_c} V_{j+1} & V_j \geq V_{j+1} + V_D \\ V_j\left(1 + \dfrac{R_{NTC}(T_j)}{R_j}\right) & V_j < V_{j+1} + V_D \end{cases} \quad (3\text{-}13)$$

If the neuron (neuron *j*) drives another neuron (neuron *j+1*) with one branch of one-directional energy transfer and, at the same time, is driven by another neuron (neuron *j-1*), as depicted in Figure 3.6 (c), one may describe the relationship as:

$$V_{in} = V_j\left(1 + \frac{R_{NTC}(T_j)}{R_j \| R_c/2}\right) - \frac{R_{NTC}(T_j)}{R_c}\left(V_{j-1} + V_{j+1}\right) \quad (3\text{-}14)$$

A general formula for describing the behaviour of a neuron (neuron *j*) with *m* connections is, only if $V_j > V_{j+(1..m)} + V_D$ for all:

$$V_{in} = V_j\left(1 + \frac{R_{NTC}(T_j)}{R_j \| R_c/m}\right) - \frac{R_{NTC}(T_j)}{R_c} \sum_{i=j}^{j+m} V_i \quad (3\text{-}15)$$

Thus, the state of the neighbouring neurons (*e.g.*, V_{j+1}) and the input signal received at each neuron ($V_{in,j}$) affects its state (V_j). As discussed and illustrated earlier, the nonlinearity comes from the temperature dependence of the thermistor's resistance, $R_{NTC}(T_j)$. The temperature of the thermistor (T_j) in the proposed neuron structure depends on the input signal, $V_{in,j}$ and the state of the neuron, V_j. To align with the definition of echo state networks, we can illustrate the equation (3-15) for neuron *j* as:

$$W_{in}^{(j)}u = V_j\left(1 + \frac{R_{NTC}(T_j)}{R_j \| R_c/2}\right) - \frac{R_{NTC}(T_j)}{R_c}\left(V_{j-1} + V_{j+1}\right) \tag{3-16}$$

Even though it seems to be hard to solve the equation (3-16), the nonlinear interactions of the neurons with each other are evident from this equation. For a reservoir of *N* neurons, the weight matrix would be of the form:

$$W_{res}{}^{(N\times N)} = \begin{bmatrix} W_{11} & \dots & W_{1N} \\ \vdots & \ddots & \vdots \\ W_{N1} & \dots & W_{NN} \end{bmatrix} \tag{3-17}$$

For a fully connected network, all the matrix elements are non-zero. However, based on the signals received at each neuron and whether the diodes are turned on or off, some or all non-diagonal elements can be zero. Thus, the weight matrix will self-adjust depending on the signals received at the neurons, either from input or neighbouring neurons.

There are two other types of coupling structures used in this chapter, nonlinear summing two states using a neuron driven with a control signal (Figure 3.7 (d)) and extraction of temporal features (Figure 3.7 (e)). Nonlinear summing, also known as nonlinear element-wise summing in computer science, is a way in which the summing of neurons can aid in feature extraction by allowing the network to learn complex, nonlinear relationships between the input data and its features.

$$V_C = V_j\left(1 + \frac{R_{NTC}(T_j)}{R_j \| R_c/2}\right) - \frac{R_{NTC}(T_j)}{R_c}\left(V_{j-1} + V_{j+1}\right) \tag{3-18}$$

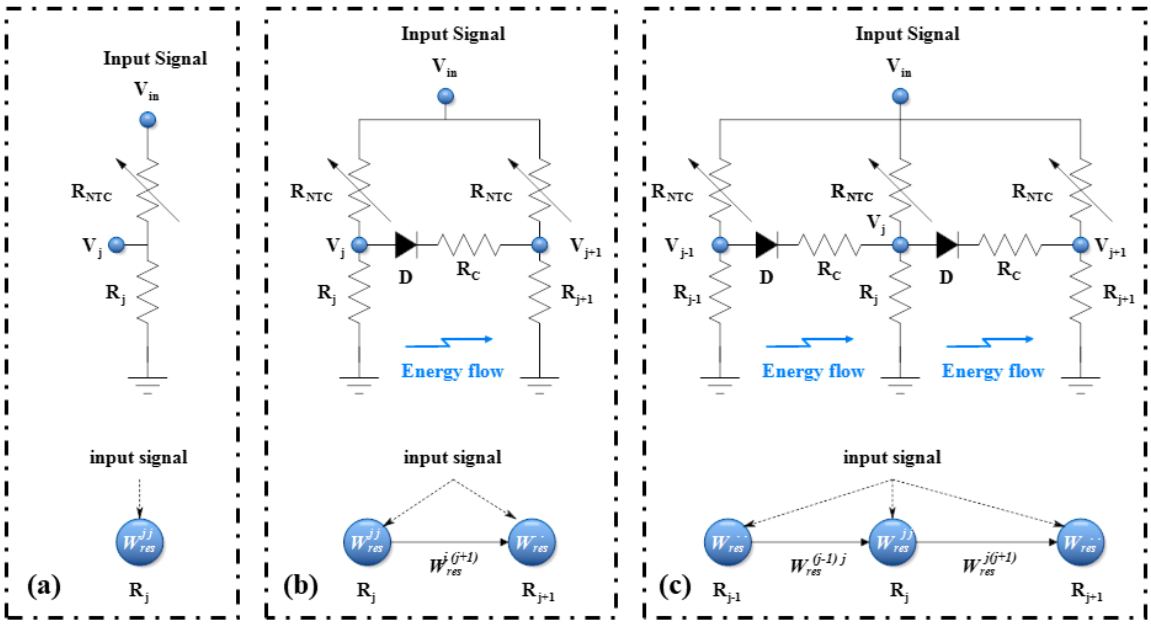


Figure 3.6. Different coupling mechanisms employed in the reservoir.

The reservoir could capture long-term dependencies in sequential data using a temporal element-wise coupling, essential for time series analysis. The ability to selectively retain or forget information from previous time steps allows the network to maintain a memory of past inputs while adapting to new input patterns.

$$V_{in,j} = V_j\left(1 + \frac{R_{NTC}(T_j)}{R_j \| R_c}\right) - \frac{R_{NTC}(T_j)}{R_c} V_{j+1}$$
$$V_{in,j+1} = V_{j+1}\left(1 + \frac{R_{NTC}(T_{j+1})}{R_{j+1} \| R_c}\right) - \frac{R_{NTC}(T_{j+1})}{R_c} V_j \quad (3\text{-}19)$$

The proposed physical reservoir is a nonlinear dynamical system in which the coupling weights between the neurons change over time with every input signal received. These nonlinear, varying coupling weights in the reservoir can be beneficial in processing signals because they can introduce nonlinear interactions between the reservoir nodes. These nonlinear interactions can lead to a more complex and dynamic response to input signals, which can be helpful in certain types of signal-processing tasks. Nonlinear interactions within the reservoir can enhance its ability to extract relevant features from input signals, thereby improving its performance in various tasks such as pattern recognition and time-series prediction.

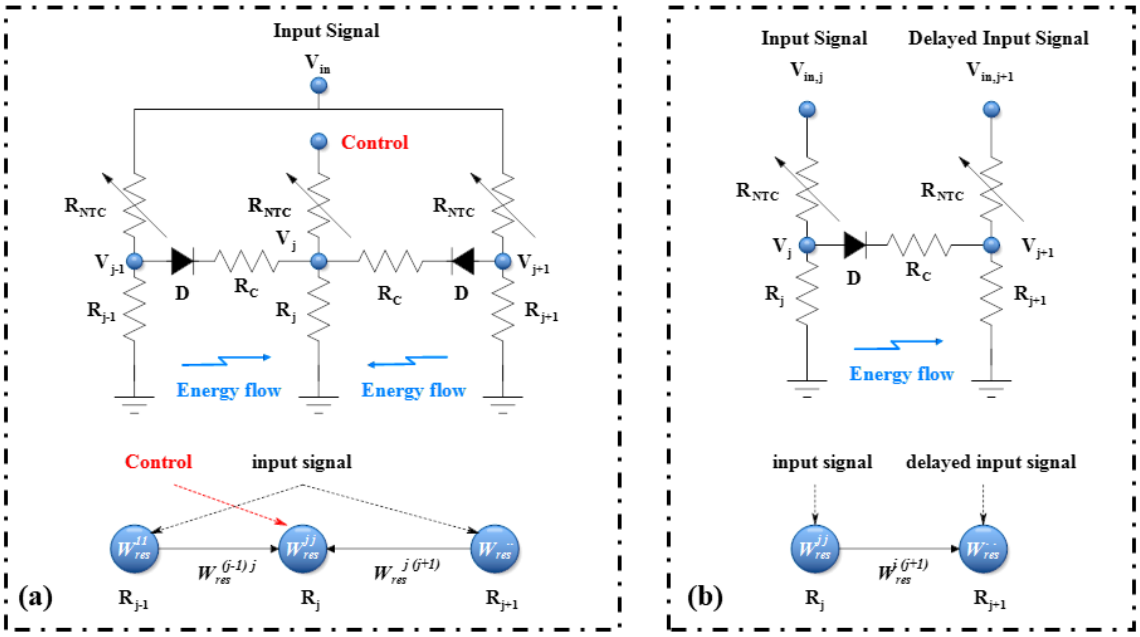


Figure 3.7. Different coupling mechanisms employed in the reservoir: (a) nonlinear summing of two states using a neuron driven with a control signal and (b) extraction of temporal features.

3.2. The Electrothermal Reservoir: The Prototype

In this study, an electrothermal reservoir developed using NTC thermistors is introduced. *Advanced Material Technologies* published this work [91], and the results demonstrate the effectiveness of this approach for generating more complex and efficient dynamics in various applications.

The Electrothermal Reservoir

The electrothermal reservoir is composed of a passive network of randomly coupled NTC thermistors. We utilized NT0515291 from Ametherm for the NTC thermistors, offering desired nonlinear characteristics. Each NTC is biased in a different but close to the nonlinear region using a bias resistance picked among 47Ω, 100Ω, 150Ω, 220Ω, 280Ω, 330Ω, and 470Ω. These bias resistances set the diagonal elements of the reservoir's fixed weight matrix, W_{res}, to the range of [0.03 0.24].

SB130-T Schottky diodes (Diodes Incorporated) and 10Ω resistors were used to implement one-directional, random electrical energy paths in the network. These connections set the off-diagonal elements of the reservoir weight matrix, W_{res}. While these

elements ought to be fixed in the reservoir, the proposed coupling offers off-diagonal weights, a nonlinear function of the input signal at each node and depends on the energy state of the two neighbouring nodes. Thus, these elements provide various off-diagonal weight values regardless of their similar and exact coupling. Another aspect is the direction of the energy flow set using these one-directional couplings, which are set to flow the energy from the highest gradient to the lowest. Additionally, almost all reservoir nodes are driven by the input signal, and some are biased using a control signal either in a self-heating mode or just below that.

Unless otherwise stated, the time step is set to 0.25*TTC, a quarter of the thermal time constant of the under-study NTC. The electrothermal reservoir was built on a regular printed circuit board (PCB), as shown in Figure 3.8. The connections between the source and measurement equipment were realized using jumper wires. The NTCs were mounted at a 0.5 cm distance from each other and 1 cm from the substrate to prevent heat convection among the reservoir nodes. Diodes are mounted to the back end of the PCB substrate to avoid the possible temperature-dependent shift in the diode characteristics due to the generated heat in the thermistors.

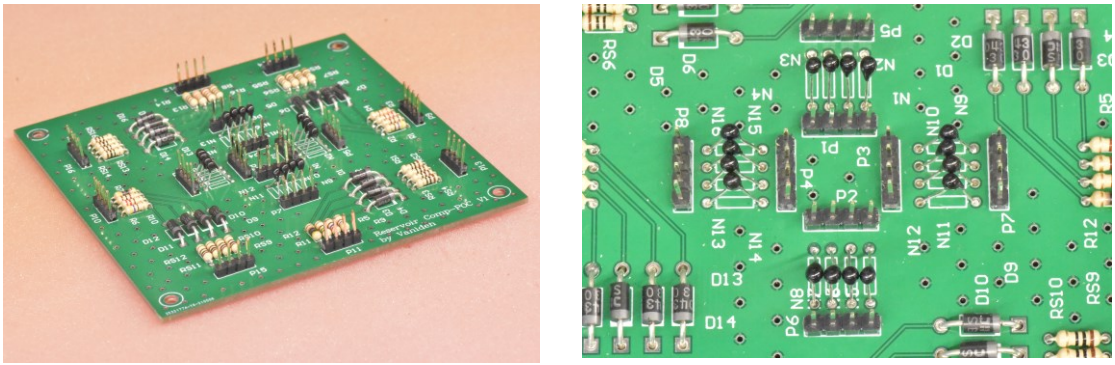


Figure 3.8. The prototype mounted on a PCB.

Figure 3.9 illustrates the realized thermo-reservoir with a size of 48, consisting of 6 sub-reservoirs with eight principal nonlinear neurons. Each sub-reservoir of size eight is driven with shifted inputs {u(t), u(t-d), …, u(t-5d)} in the range of d = {0, …, 5}, where *d* is the timestep at which the data are introduced to the reservoir.

Figure 3.10 illustrates the couplings where input or control signals can drive all or some states. Control signal biases the states, either in a self-heating mode or just below that. Each state driven by the input signal can share some of its electrical energy with the

lower state through the designated path in an ordinal way. However, when a control signal is used, the lower energy states share more of their energy rather than the states with higher energy levels, creating a highly nonlinear electrical energy exchange between the states. The states of the reservoir are depicted in Figure 3.11.

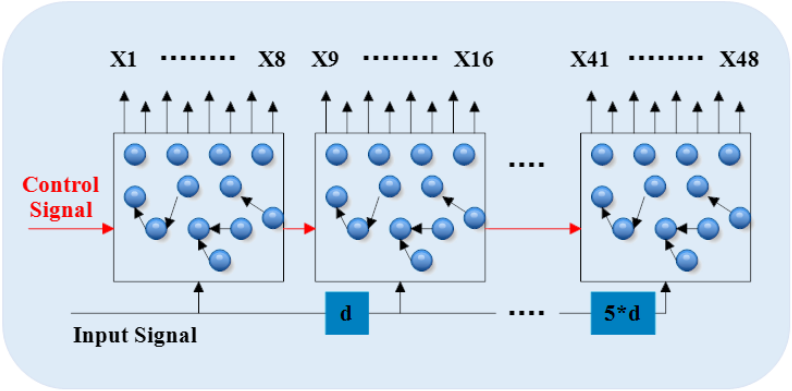


Figure 3.9. Realized thermo-reservoir consisting of 6 sub-reservoirs with eight principal nonlinear neurons each, driven by shifted inputs {u(t), u(t-d), …, u(t-5d)} in the range of d = {0, …, 5}."

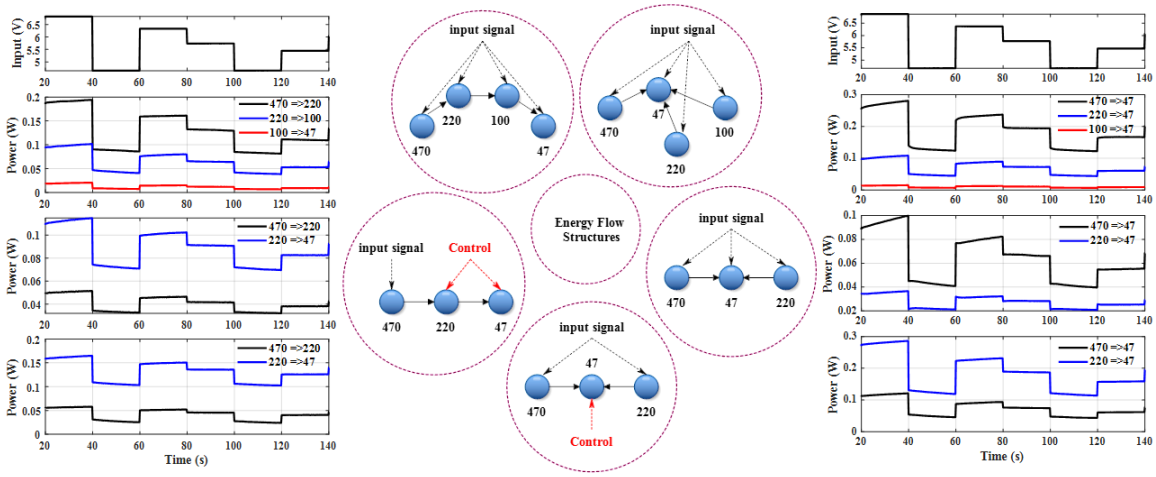


Figure 3.10. The electrical energy exchanged between coupled nodes. Each node is represented by a neuron having a distinct bias resistance value. The diagram highlights the variability in bias resistance values across the neurons by labelling the neurons by their resistance value.

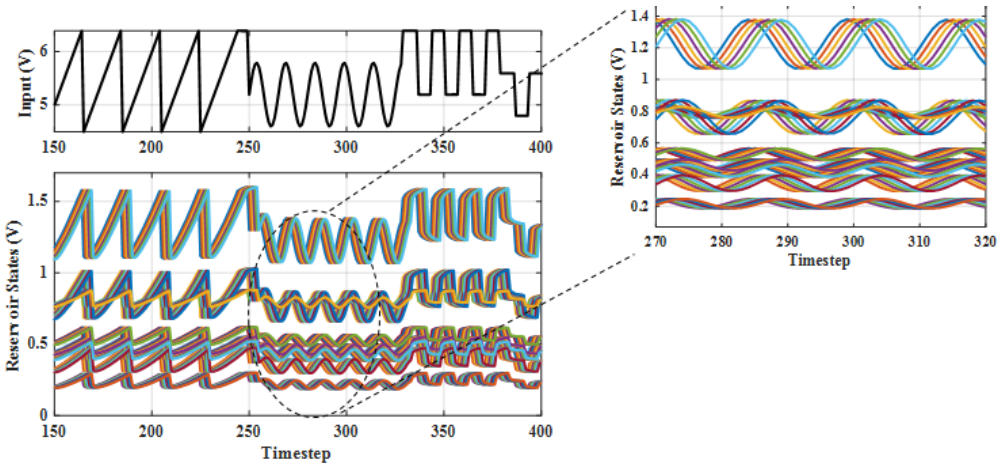


Figure 3.11. States of a rich thermo-reservoir consisting of six sub-reservoirs, each with eight nonlinear neurons. The sub-reservoirs are driven with shifted inputs in the range of 0 to 5 timesteps.

Input Layer

For physical RC, the input layer may be realized in different ways. The most straightforward approach scales the input data to the reservoir and applies it to the nonlinear nodes of the reservoir). A richer nonlinear dynamic may be created by keeping a history of the input data and applying it sequentially to the reservoir (Figure 3.12). Both these approaches may be employed for near-sensor processing applications. However, the additional computational power offered by the second approach necessitates using a sample-and-hold mechanism which adds to the system's complexity. A final approach in physical RC is to remove the input layer and build the reservoir from nonlinear, coupled sensors. The output layer then monitors the state of the reservoir and produces a signal that reflects the state of the reservoir. In this case, the reservoir state changes directly with the parameter of interest.

The input layer used in this study, as shown in Figure 3.12, consists of an analog shift register, shifting the input signal up to five timesteps, mapping six shifted, sequential signals {*u(n)*, *u(n-1)*, …, *u(n-5)*} into the reservoir.

Output Layer

The output layer runs a linear regression and decision tree implemented in a MATLAB® environment for the time-series prediction and event detection. The code is designed to apply to all the selected output signals from the reservoir (*i.e.,* 48 temporal signals).

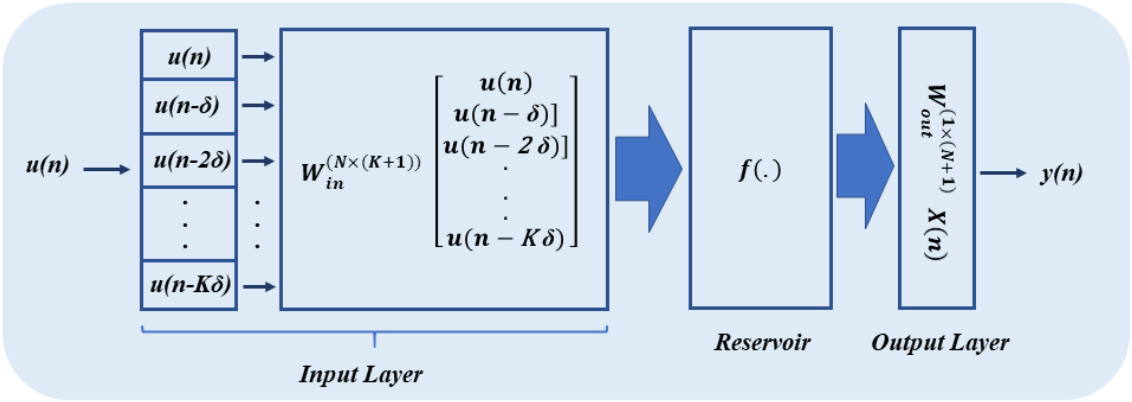


Figure 3.12. The realized input layer for the electrothermal RC.

3.3. Benchmark Analysis

The proposed electrothermal reservoir was utilized to perform two tasks. The first task involves using the reservoir to predict a times-series signal. The reservoir detected a particular event in incoming data for the second task. The data collection setup is illustrated in Figure 3.13. The input signals and output data were collected using a National Instruments PXIe-6363 I/O module through a LabVIEW interface at a rate of one sample per second. Due to the limited number of data acquisition channels (32 AI channels), I utilized an NI switch module (NI PXI-2564) to measure various signals if more than 32 neurons were employed.

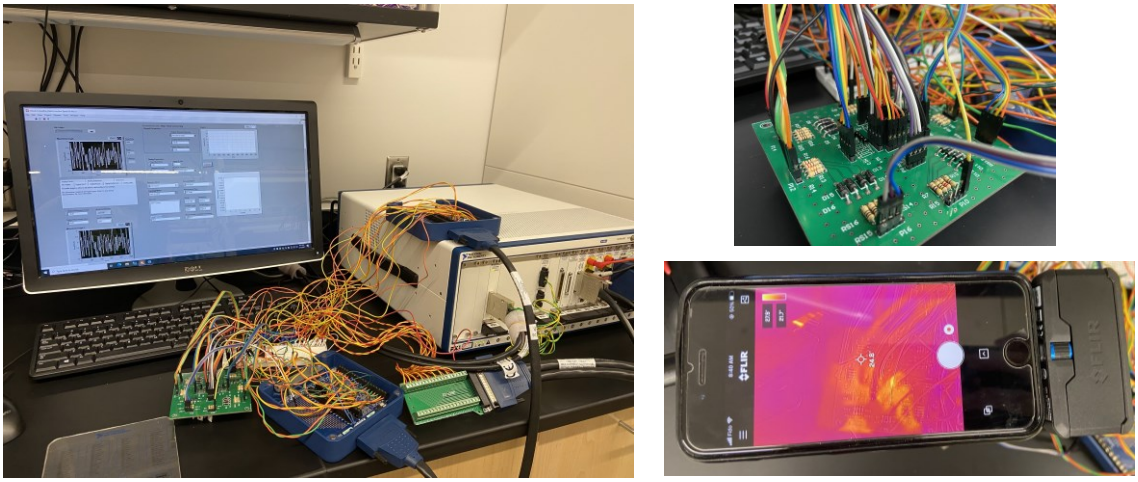


Figure 3.13. Data acquisition setup.

3.3.1. Time-Series Prediction

There are some real-life applications where continuous monitoring of a parameter, such as the concentration of specific gas species in an environment, is vital. The performance of the physical computer can be evaluated using a time-series signal to assess its capability in detecting a particular output function (e.g., concentrations) with the time history of past incidents. In this case, a standard test uses a nonlinear autoregressive moving average (NARMA) of the input signal. The input signal is a randomly distributed sinusoidal, sawtooth, and square waveform at different frequencies with amplitudes falling from 4.5V to 6.5V. NARMA is a discrete-time temporal task with an n^{th} -order time lag.

In evaluating the NARMA time series prediction, a dataset of size ~570 was used, which was split into a training set and a testing set. The training set comprised 70% of the dataset, while the remaining 30% was allocated for testing. This division allowed for the model to be trained on a substantial amount of data, which could improve its accuracy, while also ensuring that the model's performance was evaluated on a sufficiently large and diverse dataset. Figure 3.14 (a) shows the predicted and real values for various orders of nonlinearity and memory requirement (represented by n). An electrothermal reservoir with 48 nodes can compute up to a 5^{th}-order NARMA with good agreement between the actual and predicted values (see Figure 3.15 (a)). Increasing the orders of nonlinearity and memory beyond the 7^{th} order leads to increasing error. Recorded root mean squared error (RMSE) reveals that the prediction error is less than 56mV for n ={1,2,…,5}, while it experiences an increase and reaches 222mV for NARMA10.

Developing physical signal processors needs to account for the time the system needs to respond to the input. The system will ignore changes at the input much faster than its response time (i.e., fast changes are filtered). On the other hand, waiting too long for the system response to become stable increases the computational time. For the electrothermal computer, we used the NTC's thermal time constant (TTC) (i.e., the time required for the voltage across the thermistor to reach 63.3% of its final value from 0) as the measure of response time. Figure 3.15 (b) compares the system's response to inputs that vary at different time scales (0.5 TTC and 1.5 TTC). As can be seen, when the input changes quickly compared to the natural response time of the reservoir, the output retains a significant portion of the past events. The stronger dependence on past events can be balanced by using a larger reservoir exhibiting a more complex dynamic behaviour (Figure

3.15 (c)). These results demonstrate the importance of choosing the correct reservoir parameters for the accuracy and speed of computations in physical sensor computing.

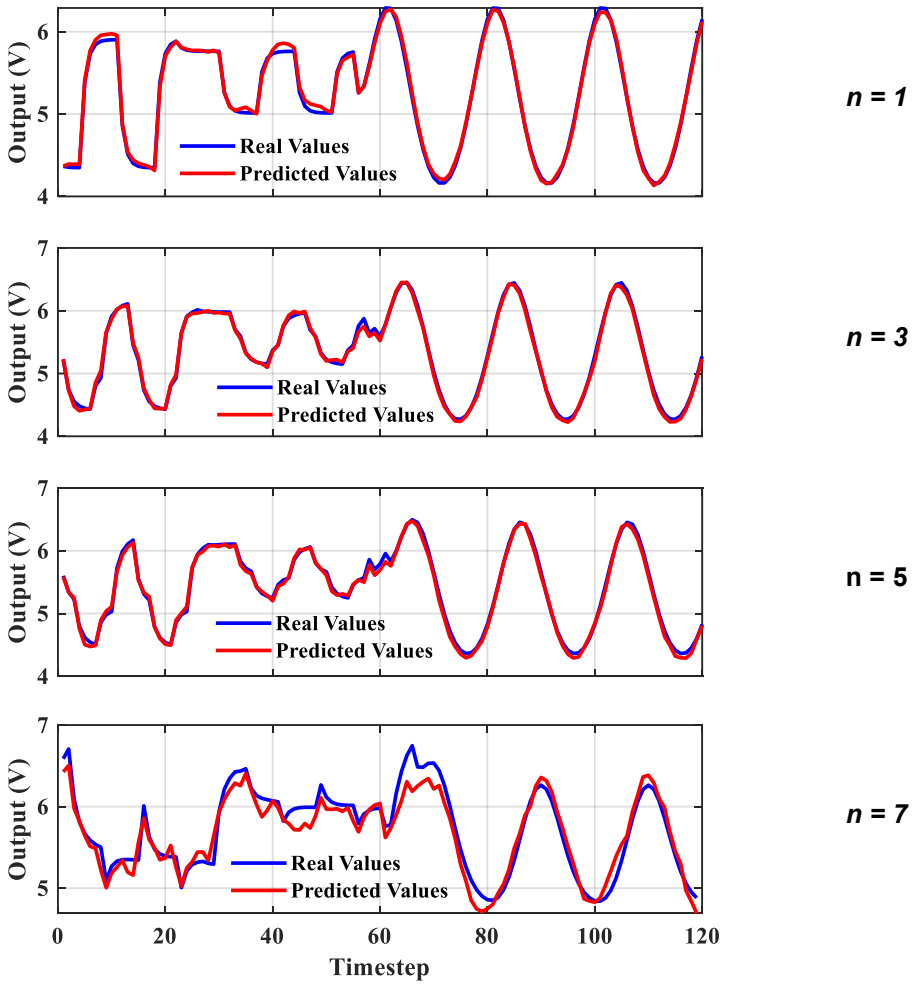


Figure 3.14. The capability of a reservoir of size 48 in computing problems with different levels of memory requirements and nonlinearities.

3.3.2. Event Detection

This experiment aimed to evaluate the capability of a physical computer, specifically an electrothermal computer, to detect a specific event based on incoming time-series data. The approach taken in this experiment could potentially be used to build event-based triggers that could wake up an intelligent system to perform specific tasks. For example, the system could be designed to detect a particular seismic activity, triggering an alert to be sent out or other action to be taken.

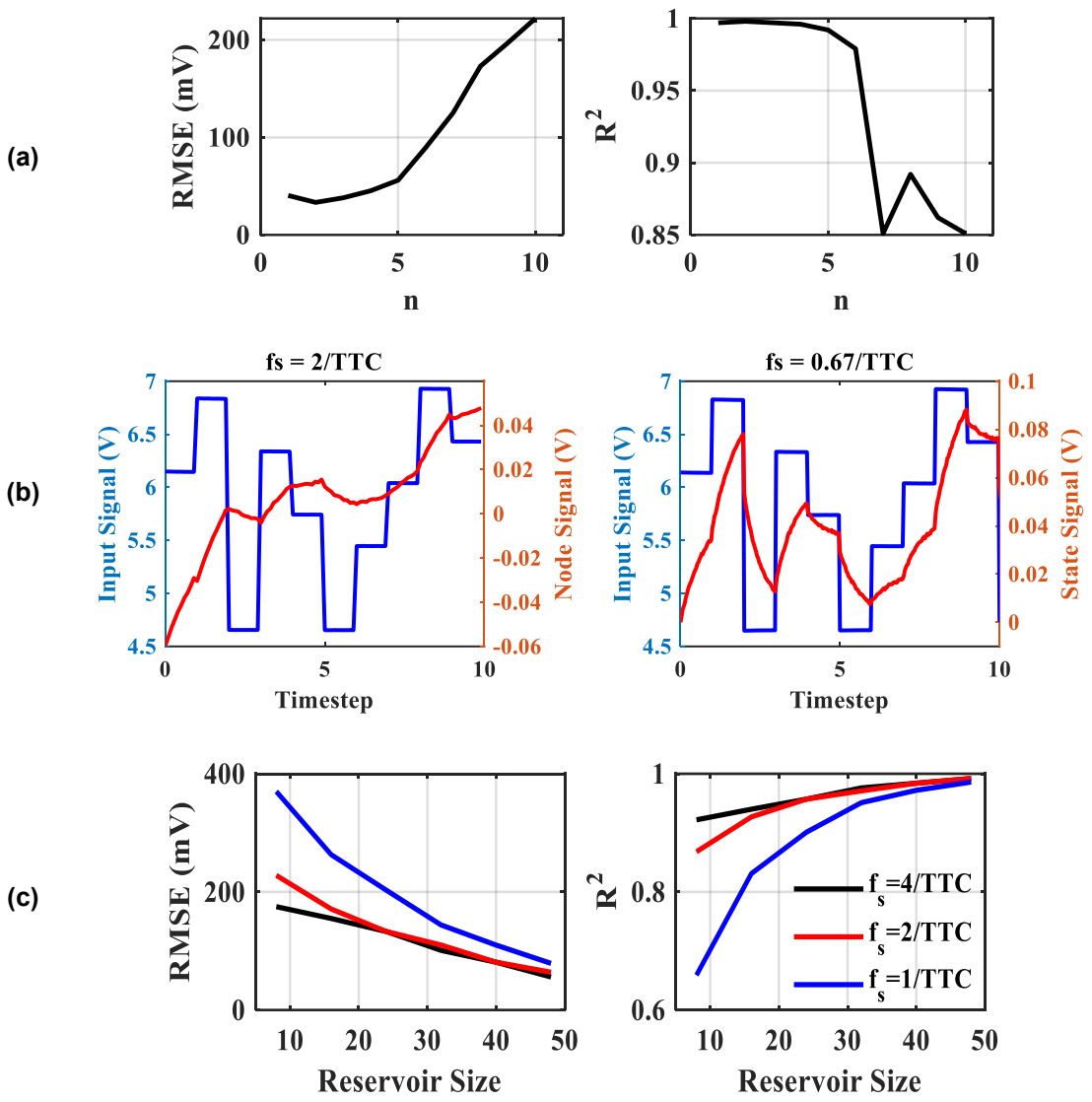


Figure 3.15. (a) RMSE and R2 of the reservoir versus various complexity levels of the under-study task (n). (b) Transient response of a reservoir node at various fs. (c) Importance of the fs on the reservoir performance.

In this experiment, the input signal was a decaying Gaussian pulse waveform distributed in various time windows with varying pulse widths. The specific events were mounted onto another pulse signal with various levels to challenge the system further. The amplitude and duration of the events changed over time, making the event-detection problem harder to solve. A limited demonstrative set of 52 specific events was introduced to the electrothermal computer. The electrothermal processor was trained to detect these events. As depicted in Figure 3.13, the experiment results showed that the electrothermal computer could detect the events with 98% sensitivity and 100% specificity to all events. In other words, the computer could correctly identify almost all events with few false

positives. We also achieved a 99% accuracy in event detection, indicating that the system was highly effective at identifying specific events. A truth table for the event detection is presented in Figure 3.17.

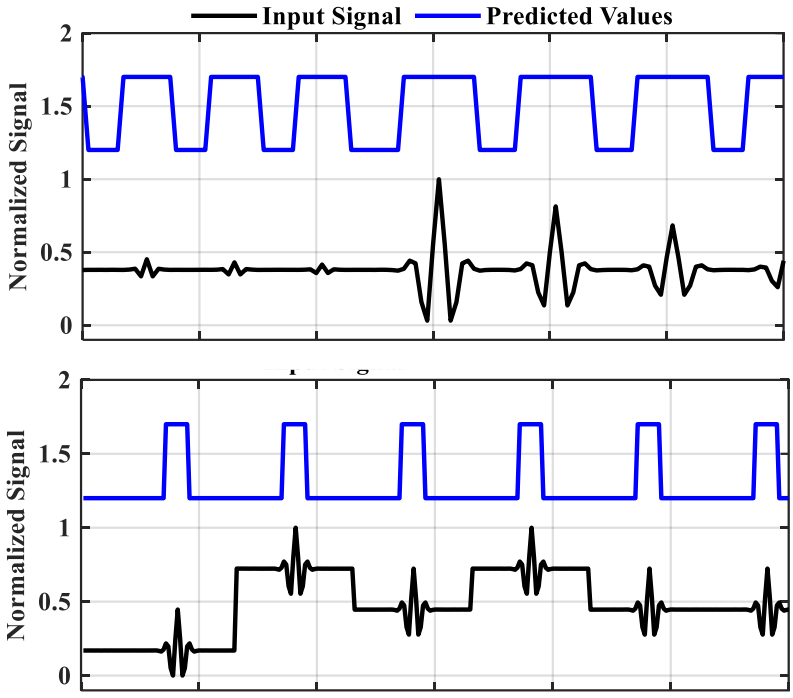


Figure 3.16. Performance of the proposed electrothermal reservoir in detecting a specific event from non-event instances.

Figure 3.17. The truth table for the specific event detection among non-event instances of the proposed electrothermal reservoir.

3.4. Summary

Physical signal processing presents a plethora of opportunities for developing specialized computers suitable for various applications. In particular, sensor signal processing can significantly benefit from this approach since, in most cases, the context produced from sensor data are more valuable than the raw data. This chapter introduced the electrothermal RC model to develop physical signal processors, focusing on cases more suitable for physical realization. The chapter also covered relevant aspects of modelling for the electrothermal RC. Similar concepts can then be applied to develop integrated physical computers using miniaturized integrated thermistors. Electrical coupling was used for creating connections within the developed reservoir in this chapter. However, since the platform operates based on the self-heating of thermistors, another potential coupling mechanism might be thermal coupling, where the generated heat is transferred from one neuron to another, as mentioned earlier in this chapter. The next chapter focuses on the development of 3D-printed reservoir computing platforms based on thermal coupling.

Chapter 4.

Exploring the Potential of 3D Printed Computing Platforms

Additive Manufacturing or 3D printing is a rapidly developing technology that enables the fabrication of complex, multi-layer, and even multi-material structures at once using various printable materials [93]. Advances in 3D printing technologies and additive manufacturing have been utilized to develop 3D-printed sensors and hence, 3D-printed intelligent systems (i.e., structures with embedded sensing capabilities) [94]. However, typical 3D-printed systems, including 3D-printed sensors, are generally passive components, lacking computational capability.

The desire to add computation capability in analog or digital domains has motivated research on developing 3D-printed transistors and active components [95]. Much of this research has thus far focused on developing transistors for analog signal processing or conventional von Neuman digital processors. However, these transistors are far from silicon devices in terms of performance and are printed at much lower densities than the existing silicon microelectronic chips, severely limiting their utility as computational elements.

Alternatively, 3D-printed neuromorphic devices and processors have been proposed to add computing power to 3D-printed structures [96]–[98]. 3D-printed optical signal processors have also been proposed as a solution to circumvent the challenges on the electrical side for processing large quantities of input data [99]. Nonetheless, the existing solutions for signal processing with 3D-printed structures face significant challenges in scaling up the computation capabilities or requiring sophisticated tools to recover the processed data.

The use of 3D printing technology in developing near-sensor computing platforms has the potential to revolutionize the field of customized intelligent system development. The ability to mass-customize processors, combined with the precise control of physical structure offered by 3D printing, allows for the creation of small and lightweight processors, making them ideal for many applications. Furthermore, 3D printing can also reduce the cost and time of production, making it more accessible for a wide range of applications.

Additionally, 3D-printed computing platforms can offer real-time processing while easing the integration of various sensors, leading to improved performance and efficiency.

The material used for 3D-printed computing platforms is critical for the device's performance. Carbon-filled Polylactic Acid (PLA) is a conductive filament that is particularly well-suited for this application. This material is a thermoplastic polymer reinforced with carbon particles, giving it high electrical conductivity. Additionally, it is a low-cost and widely available material that is easy to work with using fused deposition modelling (FDM) technology. FDM is presently the most common 3D printing technology, where a filament of the printed material is melted and deposited selectively using a printer head. Depending on the type, the nozzle temperature for printing these conductive filaments is between 120℃ to 250℃ [100].

We will introduce carbon-filled PLA as a conductive filament for 3D-printed computing platforms as an appropriate choice due to its electrical conductivity, low cost, wide availability and suitability for FDM technology. It also provides nonlinear and time-dependent responses beneficial for reservoir computing. Thus, this chapter focuses on studying the behaviour of 3D-printed neurons and reservoirs. We build a mathematical model for the 3D-printed reservoir with the mathematical weights defined based on the electrical and thermal processes involved in the reservoir. Physical phenomena involved with the thermal coupling of the neurons are described by heat transfer, and the electrical coupling follows Ohm’s law. Finally, we will compare the derived model with the experimental analysis and numerical simulation of the developed 3D-printed reservoir using COMSOL Multiphysics.

4.1. The 3D-Printed Neuron

The 3D printing technology with carbon-filled PLA as a conductive filament offers a solution for developing resistive networks for reservoir computing. These resistors are printed using FDM technology, which involves the melting and depositing of a filament of the printed material selectively using a printer head.

One of the key advantages of using 3D-printed resistors is that they exhibit significant nonlinear responses due to self-heating at relatively low temperatures (50-60℃). Figure 4.1 illustrates the reaction of a 3D-printed resistor under different conditions.

The resistors exhibit a nonlinear response under high currents due to self-heating. The resistors exhibit a time-dependent response if the time spent at each current level is shorter than the time needed to reach thermal equilibrium. For these IV characteristics, I swept the current passing through a piece of 3D printed resistor forward and backward and measured its voltage after various time intervals by Keysight 2901A precision source/measure unit (SMU).

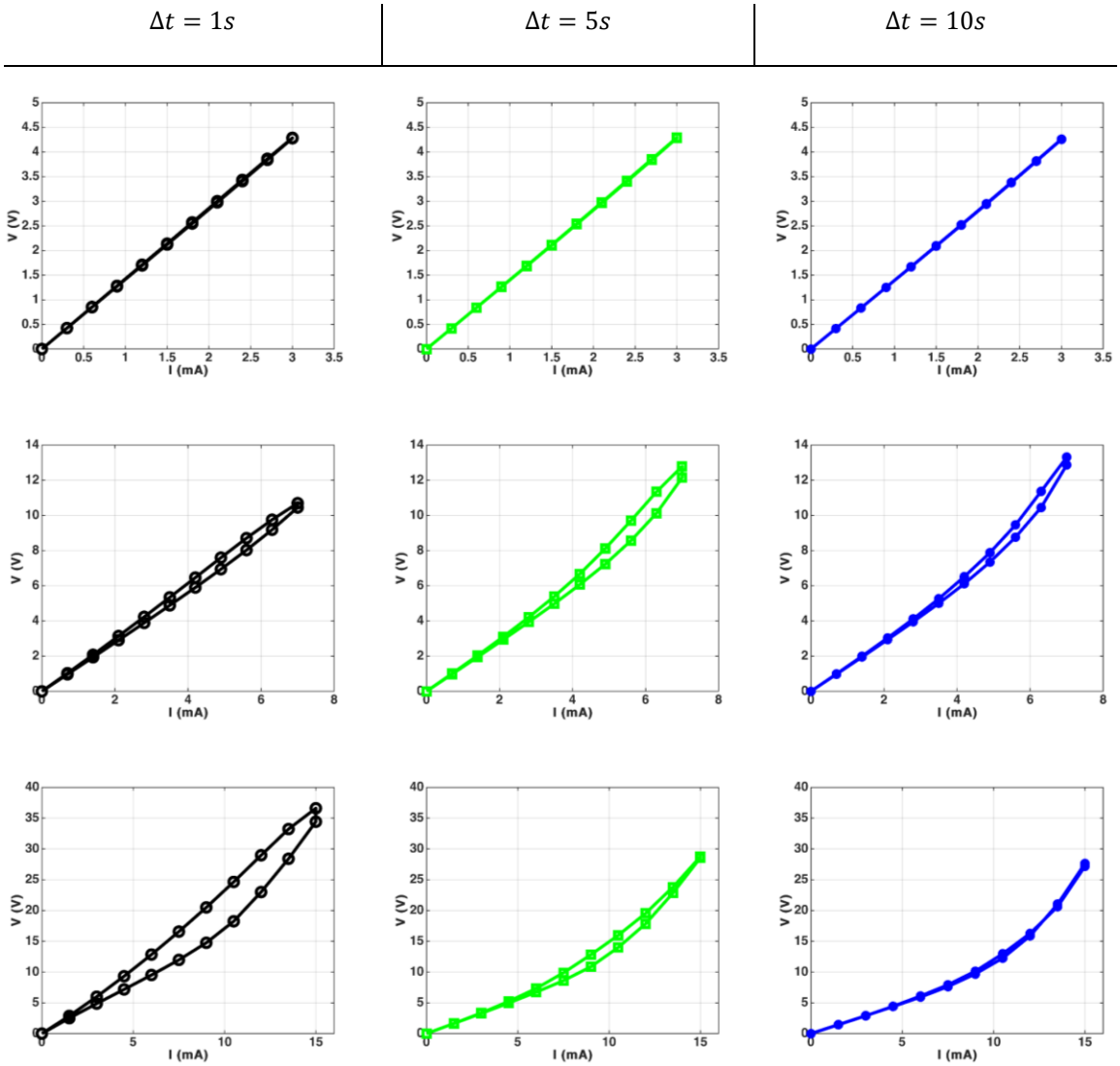


Figure 4.1. The current-voltage characteristic response of a free-hold 3D-printed resistor under different test conditions. In each case, the resistor's current was increased from zero to the maximum I_{max} and back to zero. The resistors' current was held constant for Δt (shown at the top) before measuring the voltage across the resistor and proceeding to the next step. Notably, under these test conditions, the resistor response is repeatable, indicating a permanent change to material response has not yet occurred.

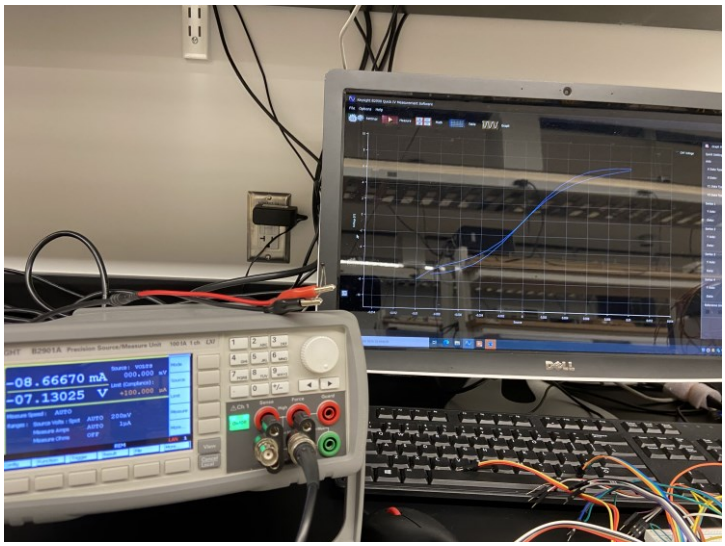
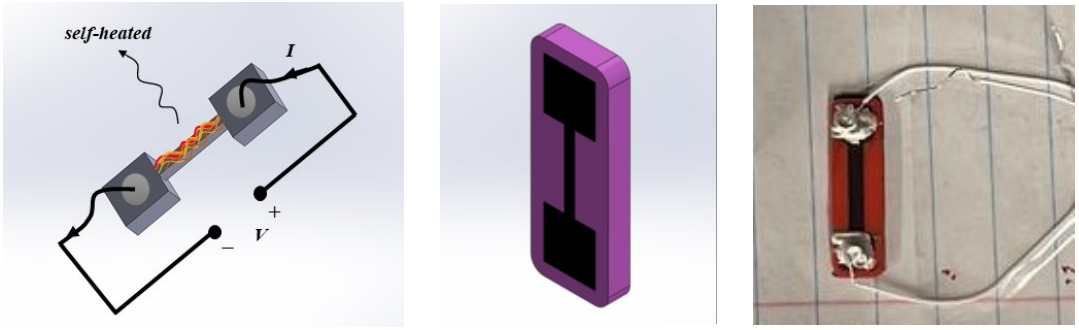


Figure 4.2. The measurement setup for IV characteristics.



(a) **(b)** **(c)**

Figure 4.3. (a) The resistor/neuron model, (b) the 3D model of a resistor created in SolidWorks where the black colour shows the resistor printed surrounded by regular PLA, and (c) the 3D-printed resistor. Note that regular PLA is available in different colours to print, while C-PLA is available in black colour. There are not significant differences in terms of thermal/mechanical properties between regular PLAs with different colours.

The setup for these measurements is shown in Figure 4.2. The nonlinearity and time dependence in the responses of these resistors makes them ideal candidates for developing physical RC systems based on ESN topology. Figure 4.3 illustrates the 3D model of a 3D-printed resistor developed in SolidWorks and its printed version.

The advancements in developing physical computing platforms promote the importance of connecting RC to physical RC to enhance our understanding of the design process. Nonlinearity in physical systems can arise from materials or devices while fading

memory may be attributed to the time required for these materials or devices to respond to a stimulus. The connections between neurons within an RC play a vital role in facilitating the exchange of information. Although the connections among neurons in a physical RC can be realized using electrical couplings, thermal coupling can be an alternative in systems employing 3D-printed temperature-sensitive elements. Heat is generated as current (*i.e.*, the input signal) passes through a 3D-printed resistor, and it can transfer from one neuron to another under specific conditions. Consequently, thermal coupling involves establishing pathways for heat transfer among neurons/resistors, with heat transfer becoming a particularly crucial aspect to investigate. In the case of a reservoir consisting of 3D-printed resistors, thermal coupling occurs when these resistors/neurons are printed close to each other, ensuring a short path for the generated heat to transfer from one resistor/neuron to another.

4.2. Analytical Model

A computational model was introduced in Section 2.1 for feedforward and recurrent neural networks. Feedforward NNs, in their most general form, are modelled using the weight matrices connecting neurons in different layers, W, the input signal u, and the response of neurons, x, *i.e.*, $f(W, u, x)$. The weights are typically time-independent and learned during training to optimize the network's performance on a specific task.

RNNs, on the other hand, are neural networks designed to process sequential or time-series data by capturing temporal dependencies. RNNs are modelled using the weight matrix connecting the previous hidden state to the current hidden state W_{hh}, the input signal $u(t)$, and the response of neurons in the previous and current time steps, $x(t-1)$ and $x(t)$, i.e., $f(W_{hh}, u(t), x(t), x(t-1))$. The weight matrices in RNNs, similar to NNs, are also time-independent. During the training process, the weights are typically learned through backpropagation through time.

In digital implementations of RC, the weight matrix is typically fixed and time-independent, similar to NNs and RNNs. Therefore, the reservoir can be modelled by $f(W_{res}, u(t), x(t), x(t-1))$. However, in physical RC, the weight matrix, or its equivalent, can be time-dependent. This time dependence arises from the dynamic properties of the physical components used to implement the reservoir. By leveraging the time-dependent nature of the physical system, physical RC systems can exhibit rich dynamics and complex

computations, which are beneficial for time-series analysis or other tasks. The time dependence in the weight matrix allows the physical system to adapt and respond to input signals, enhancing the reservoir's computational capabilities for processing time-varying data. It is important to note again that the time dependence in the weight matrix of physical RC is specific to its physical implementation and not inherent to traditional software RC. Also, the time dependence sets limits on processing capabilities.

In this context, we derive a general analytical model for 3D-printed reservoirs to gain insight into the reservoir weight matrices and design parameters required for developing an efficient 3D-printed reservoir.

4.2.1. The Analytical Model of 3D-Printed Physical RC

For 3D-printed reservoirs, the weight matrix can be derived by understanding the thermal coupling among the resistors. We can envision this scenario as neurons heating up when current passes through them (*Heat Generation*) and the generated energy transferring to the surrounding area (*Heat Transfer*). Consequently, the temperature distribution in the medium where the resistor is located changes in response to self-heating. The amount of heat transfer and temperature distribution depends on the *Boundary Conditions* of the resistor. For instance, if the resistor is printed using a single material and measured while kept in air, heat transfer to the surrounding area may occur at a negligible rate. However, suppose the resistor is printed within a solid material, like regular PLA. In this case, the heat generated in the resistor will conduct away from the resistor and transfer to the surrounding material at a higher rate compared to the air due to the higher thermal conductivity of PLA over air.

Therefore, temperature is one of the measurable quantities in the reservoir. To obtain an approximate temperature value, heat transfer problems with specific boundary conditions must be solved. Here, we begin modelling this process by

- discussing heat generation in resistors,
- providing a brief introduction to heat transfer in solids,
- defining various boundary conditions and
- deriving static and dynamic models for the reservoir.

Heat Generation in Resistors

When an electric current passes through a resistor, it undergoes resistive heating process, where the resistor generates heat, causing its temperature to rise. The amount of heat produced is directly proportional to the square of the current (I) passing through the resistor and the resistance value of the component (R), as described by I^2R. The heat transfer from the resistor to its surroundings increases with an increase in the resistor's temperature. As an illustration, when an electric current start flowing through the resistor, the resistor's temperature rises and continues to increase until steady operating conditions are reached, and the heat generation rate equals the heat transfer rate to the surroundings. The total rate of heat generation in a resistor of volume $\mathcal{V}$ can then be determined from:

$$\dot{E}_{gen} = \int_{\mathcal{V}} \dot{e}_{gen} d\mathcal{V} \qquad \text{(W)} \qquad (4\text{-}1)$$

Considering a uniform heat generation for an electric resistance heating throughout a homogeneous material, Eq. (4-1) reduces to $\dot{E}_{gen} = \dot{e}_{\text{gen}}\mathcal{V}$, where $\dot{e}_{gen}$ is the constant rate of heat generation per unit volume.

Heat Transfer Problems in Solids

The generated heat in the resistors is conducted away from the center of the resistor to its surrounding medium. In the most general case, heat transfer through a medium is three-dimensional (3D). The temperature varies along all three primary directions within the medium during the heat transfer process. However, depending on the relative magnitudes of heat transfer rates in different directions and the desired accuracy level, the problem can be reduced to one-dimensional or two-dimensional. Suppose the temperature in a solid object varies mainly in two primary directions, and the variation is negligible in the third direction. In that case, the heat transfer problem is considered two-dimensional (2D). In a one-dimensional heat transfer, the temperature varies in one direction only, and the variation of temperature and, thus, heat transfer in other directions are negligible or zero.

Heat transfer problems are solved by considering the energy balance inside the medium. Energy balance in a 3D heat transfer problem during a small time-interval Δt can be expressed as [101]:

$$\begin{pmatrix}\text{Rate of heat}\\ \text{conduction}\\ \text{at } x,y,z\end{pmatrix}-\begin{pmatrix}\text{Rate of heat}\\ \text{conduction}\\ at\ x+\Delta x,\\ y+\Delta y,z+\Delta z\end{pmatrix}+\begin{pmatrix}\text{Rate of heat}\\ \text{generation}\\ \text{inside the}\\ \text{element}\end{pmatrix}=\begin{pmatrix}\text{Rate of change}\\ \text{of the heat energy}\\ \text{content of the}\\ \text{element}\end{pmatrix}$$

Which gives the general equation of heat conduction as [101]

$$\frac{\partial}{\partial x}\left(\kappa\frac{\partial T(x,y,z,t)}{\partial x}\right)+\frac{\partial}{\partial y}\left(\kappa\frac{\partial T(x,y,z,t)}{\partial y}\right)+\frac{\partial}{\partial z}\left(\kappa\frac{\partial T(x,y,z,t)}{\partial z}\right)+\dot{e}_{gen}=mC_p\frac{\partial T(x,y,z,t)}{\partial t} \tag{4-2}$$

where κ is the thermal conductivity of the material, which is a measure of the ability of a material to conduct heat. m is the mass of the material and C_p is the specific heat capacity of the material. Heat conduction through a solid medium in a specified direction (***e.g.,*** in the x-direction) is proportional to the temperature difference across the medium and the area normal to the direction of heat transfer but is inversely proportional to the distance in that direction, which is defined by Fourier's law of heat conduction as

$$\dot{Q}_{\text{cond}}=-\kappa A\frac{dT}{dx} \tag{4-3}$$

where dT/dx is the temperature gradient. Heat flows in the direction of decreasing temperature, and thus, the temperature gradient is negative when heat is conducted in the positive x-direction.

Boundary Conditions

Heat flux and the temperature distribution in a medium depending on the surface conditions. As discussed earlier, the general heat conduction equations were developed using an energy balance on a differential element inside the medium. Thus, they remain the same regardless of the thermal conditions on the surfaces of the medium. A heat transfer problem in a medium is incomplete without a complete description of the thermal conditions at the boundary surfaces, known as the boundary conditions. Some of the most common boundary conditions are illustrated in Figure 4.4 and summarized in Table 4-1 [101].

Table 4-1. Some of the most common boundary conditions in heat transfer.

Boundary Condition	Definition
Generalized Boundary Conditions	Generally, a boundary condition can be obtained from a surface energy balance, expressed as $$\begin{pmatrix}\text{Heat transfer}\\ \text{to the surface}\\ \text{in all modes}\end{pmatrix}=\begin{pmatrix}\text{Heat transfer}\\ \text{from the surface}\\ \text{in all modes}\end{pmatrix}$$
Specified Temperature Boundary Condition	The specified temperature boundary conditions can be expressed as $$T(0,t)=T_1 \quad T(L,t)=T_2$$
Specified Heat Flux Boundary Condition	When energy interactions at a surface are known, the rate of heat transfer (*i.e.*, the heat flux $\dot{q}$) on that surface can be used as one of the boundary conditions. $$\dot{q}=-\kappa\frac{\partial T}{\partial x}=\begin{pmatrix}\text{Heat flux in the}\\ \text{positive } x- \text{ direction}\end{pmatrix}$$
Special Case I: Insulated Boundary	The heat transfer would be zero at an insulated surface $$\frac{\partial T(L,t)}{\partial x}=0$$
Special Case II: Thermal Symmetry	For a large plate of thickness L suspended vertically in the air with its two surfaces exposed to the same thermal conditions, symmetry in the temperature distribution about the center plane, $x=L/2$, results in no heat flow across the center plane. Therefore, the center plane can be treated as an insulated surface (zero heat flux boundary condition) $$\frac{\partial T(L/2,t)}{\partial x}=0$$
Convection Boundary Condition	In practice, since most heat transfer surfaces are exposed to an environment with a specific temperature, the most common boundary condition is convection, which is defined as $$\begin{pmatrix}\text{Heat conduction}\\ \text{at the surface in a}\\ \text{selected direction}\end{pmatrix}=\begin{pmatrix}\text{Heat convection}\\ \text{at the surface in}\\ \text{the same direction}\end{pmatrix}$$ For one-dimensional heat transfer in the *x*-direction, the convection boundary conditions at the surface of $x=0$ can he expressed as $$-\kappa\frac{\partial T(0,\mathrm{t})}{\partial x}=h[T_\infty-T(0,\mathrm{t})]$$ where h and T_∞ are the convection heat transfer coefficient and temperature far from the surface, respectively.
Interface Boundary Conditions	If an object consists of multiple layers of different materials, the heat transfer problem must be solved in each layer considering the boundary conditions at each interface. At the interface of two layers, *A* and *B*, with perfect contact at $x=x_0$, (1) the two layers must have the same temperature at the contact area $$T_A(x_0,t)=T_B(x_{0,},t)$$ and (2) energy is not stored at the contact area (same heat flux on the two sides of the interface). $$-\kappa_A\frac{\partial T_A(x_0,t)}{\partial x}=-\kappa_B\frac{\partial T_B(x_0,t)}{\partial x}$$ where κ_A and κ_B are the thermal conductivities of the layers A and B, respectively.

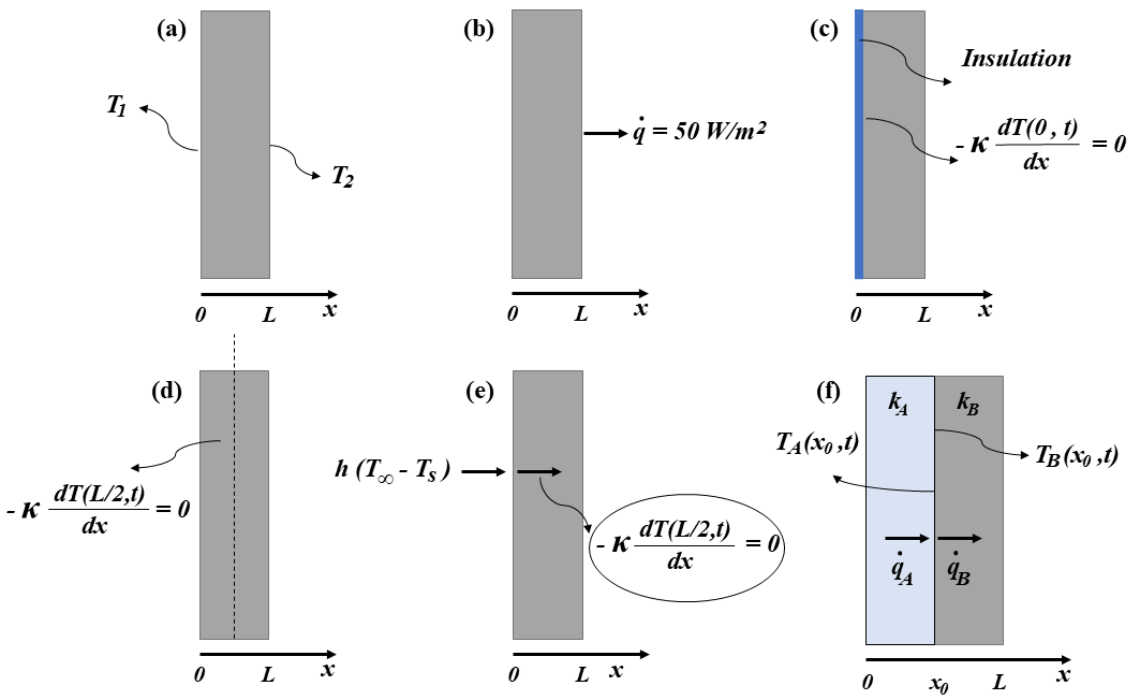


Figure 4.4. **(a) Specified temperature boundary conditions. (b) Specified heat flux boundary conditions. (c) A large plate with insulation. (d) Thermal symmetry boundary condition. (e) Convection boundary condition. (f) Boundary conditions at the interface of two bodies in perfect contact.**

4.2.2. The Static Response of a Reservoir

Here, we derive an analytic model for the static response of a reservoir consisting of four 3D-printed resistors (i.e., neurons). Consider the resistors are printed in a medium of regular PLA, as shown in Figure 4.5. For simplicity in this section, we will assume the resistors are printed with long (*L*) cylindrical cross-sections with a radius of r_i. We will solve the heat transfer problem for the medium surrounding the resistors to develop the weight matrix for the reservoir. We will consider that the temperature of the resistors does not change significantly inside it, so the average temperature for each resistor is equal to the temperature at its contact surface with the regular PLA, *i.e.,* $T(r_i)$ (See Appendix A). The input signal to the neuron/resistor is a current density ($\frac{A}{m^2}$), neuron's response is calculated as its temperature (*K*), the neurons are electrically insulated, and for simplicity, the electrical conductivity of the neurons is considered to be insensitive to temperature (no nonlinearity introduced). We want to model the static response of thermally-coupled neurons when subjected to an input signal *J*. The main focus is to gain insights into the

thermal coupling between these neurons, including their strength and the factors that influence the strength of these connections.

The first step involves deriving the temperature distribution inside a regular PLA resulting from the heating of the resistors. This entails solving the heat transfer problem within the regular PLA to determine how heat spreads. In the subsequent step, the temperature at the location of each neuron/resistor is calculated, taking into account the influence of all other neurons. Since the medium is assumed to be linear, the principle of superposition can be employed to determine the temperature distribution. In this approach, one neuron is considered to heat the environment while the others remain off, and the temperature distributions resulting from each neuron's contribution are summed. By performing these calculations, the study aims to uncover the relationship between the input signal, ***J***, the temperature distribution within the PLA, ***T***, and the resulting response of the thermally-coupled neurons, T_1, T_2, T_3, and T_4. It seeks to shed light on the strength of thermal coupling between the neurons and identify the key parameters that affect the coupling between the neurons.

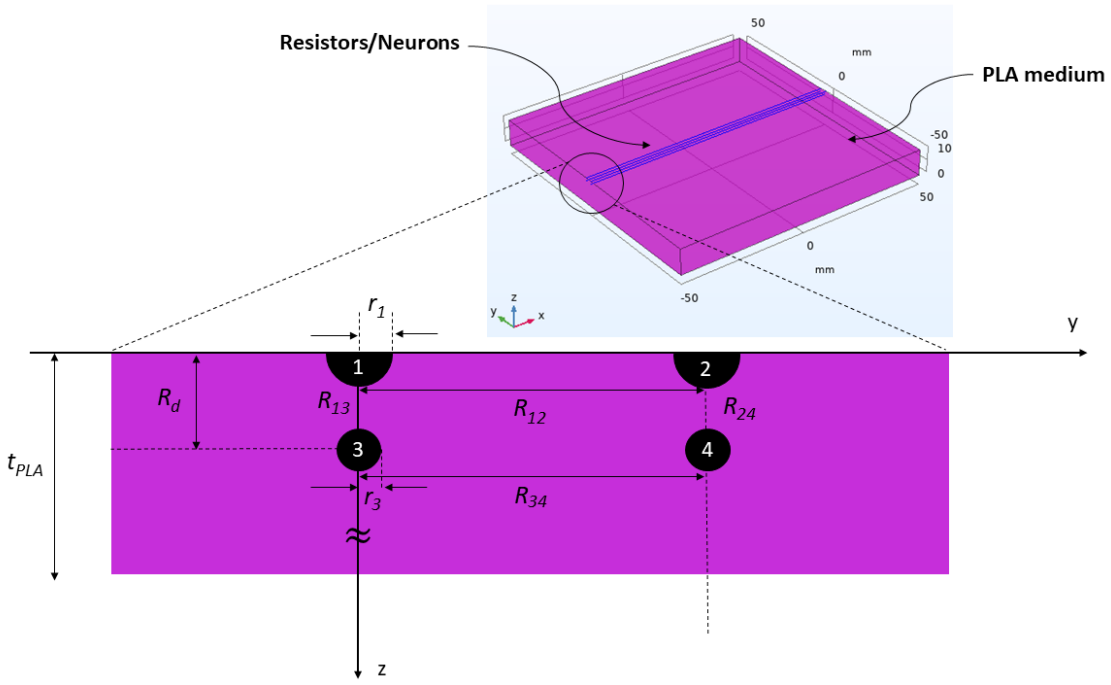


Figure 4.5. The heat transfer problem with four resistors/neurons. Resistors are long cylinders with small radii compared to medium thickness.

For the resistors, the temperature distribution will be achieved by solving the general heat conduction problem in a cylindrical coordinate system as [101]

$$\frac{1}{r}\frac{\partial}{\partial r}\left(\kappa_{PLA} r \frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \varphi}\left(\kappa_{PLA} r \frac{\partial T}{\partial \varphi}\right) + \frac{\partial}{\partial z}\left(\kappa_{PLA}\frac{\partial T}{\partial z}\right) + \dot{e}_{gen} = \rho_m C_p \frac{\partial T}{\partial t} \quad (4\text{-}4)$$

where κ_{PLA} is the thermal conductivity of the PLA material and ρ_m is the density of the PLA. The regular PLA considered in this study has large dimensions, allowing us to assume that at a location far from the resistors, the temperature reaches the ambient temperature denoted as T_a. This assumption leads to the imposition of a constant temperature boundary condition, where $T(r = t_{PLA}) = T_a$. To solve the heat transfer problem within the regular PLA, a specific heat flux boundary condition is considered at the interface between the resistor and the regular PLA. Mathematically, this condition is expressed as $-\kappa_{PLA}\frac{dT}{dr} = q$, where q is the heat flux at the contact point of the neuron and the PLA due to the heat generated in the resistor and conducted away from it to the contact surface. According to the discussions in Appendix A, the heat flux for each resistor with the radius of r_i is given as $q_i = \rho_e {J_i}^2 r_i$, where ρ_e represents the electrical resistivity.

Furthermore, if we assume that heat transfer ceases at the top surface of the PLA, where it is in contact with the surrounding air, a thermal isolation boundary condition can be applied. It is important to note that thermal isolation at a specific point can also be introduced due to the system's symmetry. Thus, we can solve the general heat conduction problem for a fully cylindrical structure instead of a half-cylindrical problem, as shown in Figure 4.6. Neglecting the dynamic part and considering that there is no heat generation within the regular PLA, the heat conduction problem described by Eq. (4-4) can be simplified. This simplification reduces the problem to focusing solely on the heat conduction process within the PLA and its interaction with the resistors.

$$\frac{1}{r}\frac{d}{dr}\left(\kappa_{PLA} r \frac{dT}{dr}\right) = 0 \quad (4\text{-}5)$$

These boundary conditions and the assumptions made allow for the formulation of a simplified heat conduction problem within the regular PLA. This simplified problem omits the dynamic aspects and considers a scenario where heat is only conducted through the PLA material without any internal heat generation. By solving this reduced problem, valuable insights can be gained regarding the temperature distribution and heat transfer dynamics within the regular PLA, providing a foundation for understanding the behaviour of the thermally-coupled neurons in the system.

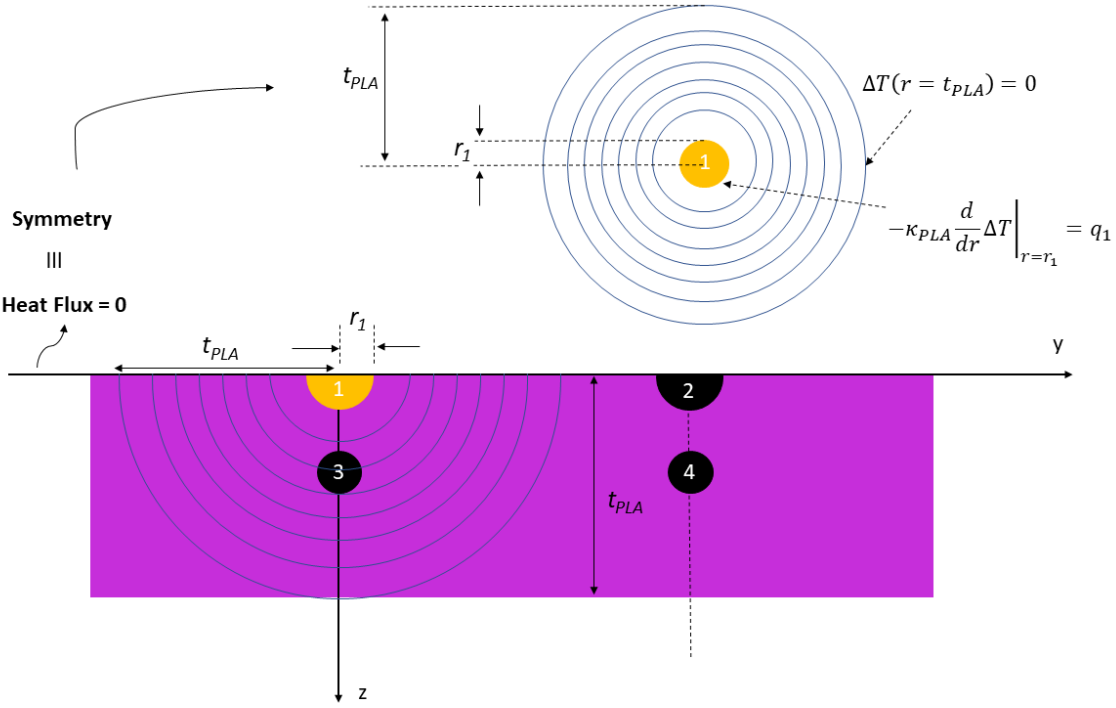


Figure 4.6. Simplification of heat transfer for resistor/neuron 1. It is assumed that the heat transfer ceases at the top surface of the PLA, where it contacts the surrounding air, allowing to application of a thermal isolation boundary condition. Thermal isolation at a specific point can be caused due to the system's symmetry. Therefore, instead of solving a half-cylindrical problem, the general heat conduction problem for a fully cylindrical structure would result in the temperature distribution due to the self-heating of resistor 1.

Solving this equation for resistor 1 considering $-\kappa_{PLA}\frac{dT}{dr}\Big|_{r=r_1} = q_1$, gives the temperature distribution for $r_1 < r < t_{PLA}$ as

$$\Delta T_1(r) = \frac{\rho J_1^2 r_1^2}{\kappa_{PLA}} \mathrm{Ln}\left(\frac{t_{PLA}}{r}\right) \tag{4-6}$$

where $\Delta T_i(r) = T_i(r) - T_a$. Note that the $T_1(r = r_1)$ gives an approximate temperature of resistor 1. For resistor 3 in the regular PLA, a thermal isolation boundary condition is introduced by considering a projection of resistor 3 along the negative *z*-axis, as illustrated in Figure 4.7. This ensures that the thermal isolation due to the surface with direct contact with air is properly accounted for. Additionally, it is essential to note that the area through which heat transfers from resistor 3 and its projection to the surrounding area differs from the previous case. Consequently, the heat flux boundary condition for resistor 3 and its

projection is modified as $-\kappa_{PLA}\frac{dT}{dr}\Big|_{r=r_3} = \rho_e {J_3}^2\, r_3$ and $-\kappa_{PLA}\frac{dT}{dr'}\Big|_{r'=r_3} = \rho_e\, {J_3}^2\, r_3$. Thus solving Eq. (4-5) for $r_3 < r, r' < t_{PLA}$ results in

$$\Delta T_3(r) = \frac{\rho_e\, J_3^2\, r_3^2}{\kappa_{PLA}} \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{r\, r'}\right) \tag{4-7}$$

where R_d represents the distance of the neuron 3 from the top surface. Now, by going back to the Cartesian coordinate system, we can find the temperature distribution for other resistors. We will keep the Cartesian coordinate system's origin at the center of resistor 1; therefore we replace $r = \sqrt{y^2 + z^2}$ in Eq. (4-6). Whereas, for resistor 3, we replace $r = \sqrt{y^2 + (z - R_d)^2}$ and $r' = \sqrt{y^2 + (z + R_d)^2}$. These changes result in (for $r_1 < \sqrt{y^2 + z^2} < t_{PLA}$)

$$\Delta T_1(y, z) = \frac{\rho_e\, J_1^2\, r_1^2}{\kappa_{PLA}} \mathrm{Ln}\left(\frac{t_{PLA}}{\sqrt{y^2 + z^2}}\right) \tag{4-8}$$

And for $r_3 < \sqrt{y^2 + (z - R_d)^2} < t_{PLA}$

$$\Delta T_3(y, z) = \frac{\rho_e\, J_3^2\, r_3^2}{\kappa_{PLA}} \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{\sqrt{(y^2 + (z - R_d)^2)(y^2 + (z + R_d)^2)}}\right) \tag{4-9}$$

A reservoir consisting of four resistors/neurons, as shown in Figure 4.5 with the specific dimensions provided in Table 4-2, was simulated using COMSOL Multiphysics® 5.5 software. Several physics modules must be combined to simulate and model self-heating in a resistor. Two physics were employed here: electric currents (ec) from the AC/DC module and heat transfer in solids from the Heat Transfer module, and a coupled physics (*i.e.,* Multiphysics) known as electromagnetic heating to study the effect of self-heating in the under-study electrothermally conductive material (i.e., the conductive PLA). Readers are referred to Appendix A for more information on the material properties used for the simulations and the specific boundary conditions. The current densities injected into the resistors/neurons were set to $J_1 = J_3 = \frac{10\ \mu A}{\pi r_3^2} = 0.127\ \left(\frac{\mu A}{(\mu m)^2}\right)$.

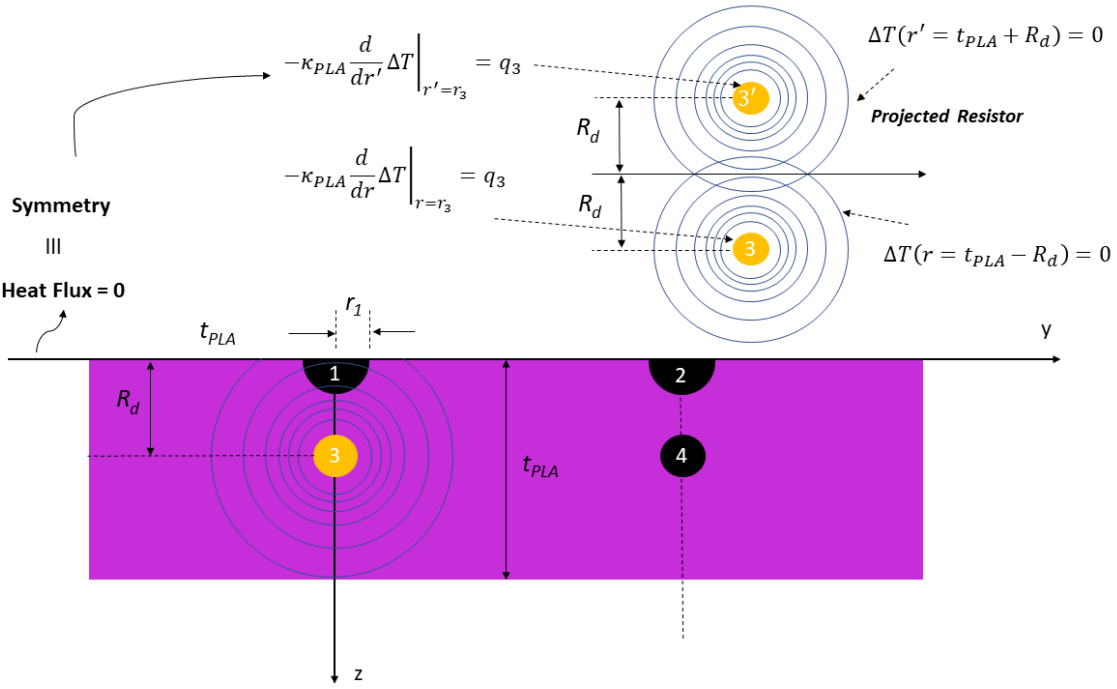


Figure 4.7. Simplification of heat transfer for Resistor/Neuron 3. To accurately represent the thermal isolation boundary condition at the top surface in contact with the air, we can consider a projection of resistor 3 along the negative z-axis.

Table 4-2. Specific dimensions used for the COMSOL MultiPhysics Simulations.

Parameter	Definition	Value
r_1 & r_2	Radius of resistors/neurons in the top layer	$5\sqrt{2}$ [μm]
r_3 & r_4	Radius of resistors/neurons in the second layer	5 [μm]
t_{PLA}	The thickness of the 3D-printed piece PLA	10 [mm]
W_{PLA}	Width of the 3D-printed piece PLA	100 [mm]
L_{PLA}	Length of the 3D-printed piece and the resistors/neurons	100 [mm]
R_{12} & R_{34}	Shortest spacing between two neighboring resistors/neurons in one layer	2 [mm]
R_{13} & R_{24}	Shortest spacing between two neighboring resistors/neurons in two different layers	2 [mm]

The simulation results were compared to those of the derived models in Equations (4-8) and (4-9). The findings are presented in Figure 4.8. The analytical model describing temperature distributions when current flows through each resistor aligns closely with the developed analytical model. Furthermore, isothermal surfaces (*i.e.,* surfaces with uniform temperature) exhibit a strong agreement with the simulation results, affirming a high level of harmony between the analytical model and the simulation outcomes. The isothermals closely match the simulations, particularly at elevated temperatures near the resistors.

However, it is worth noting that due to the dimensions extending in directions other than depth (W_{PLA} & L_{PLA} >> t_{PLA}), the isothermals may exhibit a slightly larger expansion than anticipated in these directions.

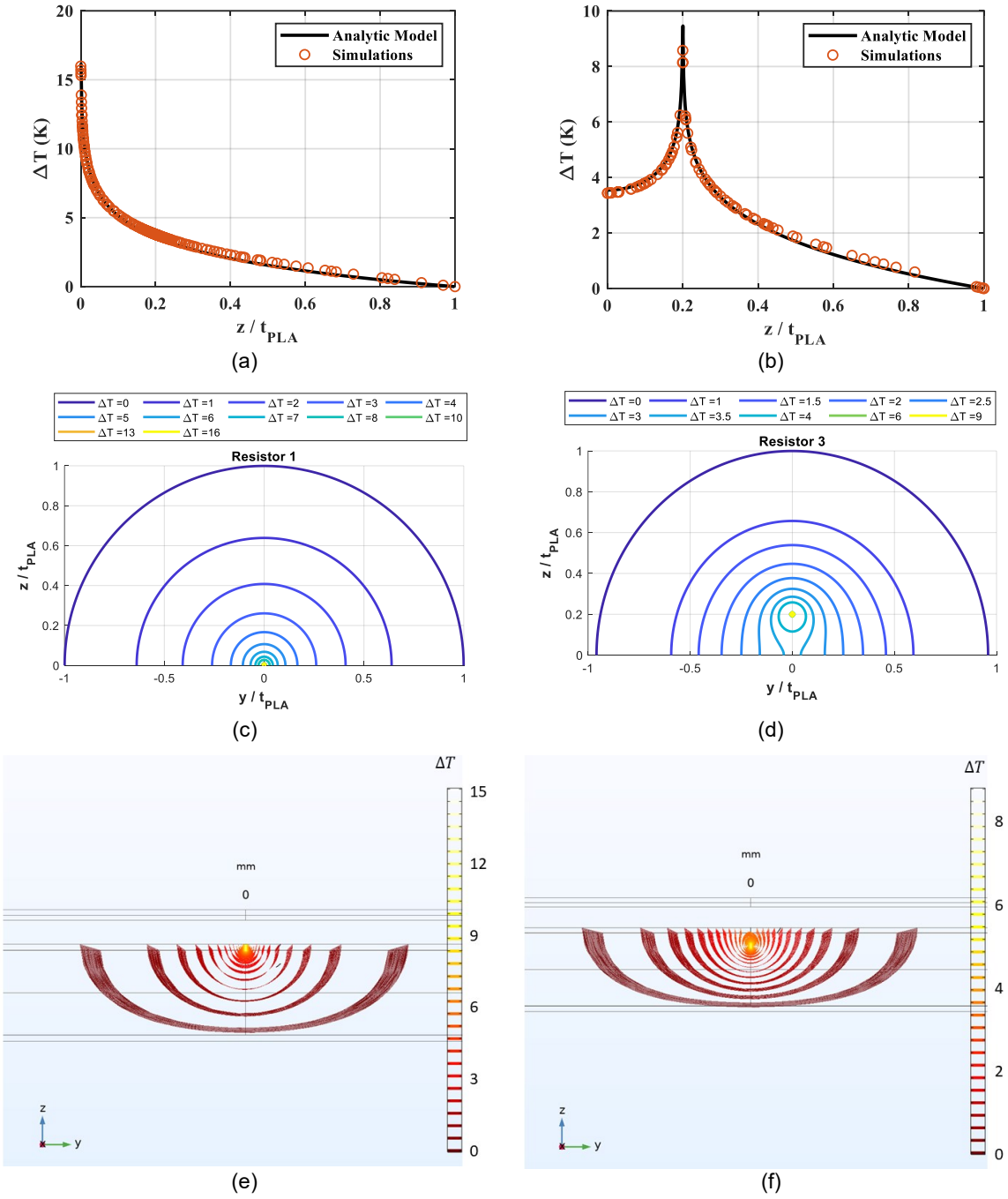


Figure 4.8. **Comparing analytic model and simulations for resistors 1 and 3. ΔT of (a) resistor 1 and (b) resistor in *z* direction when *y=0*. (c) Simulated isothermal surfaces for resistor 1 compared to its analytic model (d). (e) Simulated isothermal surfaces for resistor 3 compared to its analytical model (f).**

Now that we have solved the problem for two types of temperature distribution inside a material, *i.e.,* neurons located on the top layer and buried at any depth, they can describe the temperature distribution of the rest of the resistors/neurons in the reservoir. Therefore, for resistors 2 and 4, by replacing $y = y - R_{12}$ and $y = y - R_{34}$ the temperature distribution for resistor 2 in $r_2 < \sqrt{(y - R_{12})^2 + z^2} < t_{PLA}$ is given as:

$$\Delta T_2(y,z) = \frac{\rho_e J_2^2 r_2^2}{\kappa_{PLA}} \mathrm{Ln}\left(\frac{t_{PLA}}{\sqrt{(y - R_{12})^2 + z^2}}\right) \tag{4-10}$$

For resistor 3 is given as (for $r_3 < \sqrt{(y - R_{34})^2 + (z - R_d)^2} < t_{PLA}$):

$$\Delta T_4(y,z) = \frac{\rho_e J_4^2 r_4^2}{\kappa_{PLA}} \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{\sqrt{((y - R_{12})^2 + (z - R_d)^2)((y - R_{34})^2 + (z + R_d)^2)}}\right) \tag{4-11}$$

Superposition then implies that the temperature distribution can be achieved by summing over the temperature distribution of all resistors/neurons.

$$\Delta T(y,z) = \sum_{i=1}^{4} \Delta T_i(y,z) \tag{4-12}$$

Now, with the equation described above, we can calculate the temperature of each resistor or neuron by integrating their respective locations into the equation. This results in the static model of the reservoir, which is represented as follows

$$\begin{bmatrix} \Delta T_1 \\ \Delta T_2 \\ \Delta T_3 \\ \Delta T_4 \end{bmatrix} = W_{res}^{(4\times4)} \cdot \begin{bmatrix} J_1^2 \\ J_2^2 \\ J_3^2 \\ J_4^2 \end{bmatrix} \tag{4-13}$$

where $\Delta \boldsymbol{T}$ is the vector describing the neuron responses, $\boldsymbol{J}.^{\boldsymbol{2}}$is the input signal, and W_{res} is the weight matrix of the neural network. Considering $r_i \ll R_{ij}$ where $i \neq j$, this weight matrix is given as

$$W_{res} = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix}$$

$$= \frac{\rho_e}{\kappa_{PLA}} \begin{bmatrix} {r_1}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{r_1}\right) & {r_2}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{R_{12}}\right) & {r_3}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{{R_{13}}^2}\right) & {r_4}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{{R_{14}}^2}\right) \\ {r_1}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{R_{12}}\right) & {r_2}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{r_2}\right) & {r_3}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{{R_{23}}^2}\right) & {r_4}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{{R_{24}}^2}\right) \\ {r_1}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{R_{13}}\right) & {r_2}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{R_{23}}\right) & {r_3}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{2\, r_3 R_d}\right) & {r_4}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{R_{24} {R_d}'}\right) \\ {r_1}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{R_{14}}\right) & {r_2}^2 \mathrm{Ln}\left(\frac{t_{PLA}}{R_{24}}\right) & {r_3}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{R_{34} {R_d}'}\right) & {r_4}^2 \mathrm{Ln}\left(\frac{{t_{PLA}}^2 - {R_d}^2}{2\, r_4 R_d}\right) \end{bmatrix} \quad (4\text{-}14)$$

where $R_{12} = R_{34}$, $R_d = R_{13} = R_{24}$, $R_{14} = R_{23} = \sqrt{{R_{12}}^2 + {R_d}^2}$, and ${R_d}' = \sqrt{{R_{34}}^2 + 4{R_d}^2}$. The unit of the weight matrix is $\frac{\Omega.K}{W} m^4$. The weight matrix is contingent upon the spacing between the resistors/neurons, represented as R_{ij}. As the distance between neurons increases, their connection becomes more tenuous; conversely, bringing them closer results in stronger connectivity. Several material characteristics can influence the weight matrix, such as the thermal conductivity of the PLA—the material encasing the resistors/neurons. Another parameter is the resistivity of the resistors/neurons. A temperature-independent resistivity yields a linear reservoir matrix, while nonlinear temperature-dependent resistivity gives rise to nonlinear interactions among the resistors/neurons. A pivotal aspect in reservoir design entails incorporating diverse weight elements, which can be actualized by introducing distinct resistors/neurons with varying radii. Therefore, by manipulating the dimensions of the resistor, it is possible to create complex temperature distributions that can be utilized as part of a larger system.

To further validate our analytical model, we compared it to the simulation results for the temperature distribution within the reservoir, as outlined in Equation (4-12). This comparison, illustrated in Figure 4.9, revealed a consistent temperature distribution in both the *y* and *z* directions, further affirming our analytical model's reliability. Subsequently, the weight elements were graphed in Figure 4.10 to observe the impact of spacing between neurons, denoted as R_{ij}. As anticipated, the coupling between the neurons depends on their separation. It is noteworthy that the similarity in spacings contributes to a reduction in network complexity, owing to the presence of analogous weight elements.

(a)

(b)

(c)

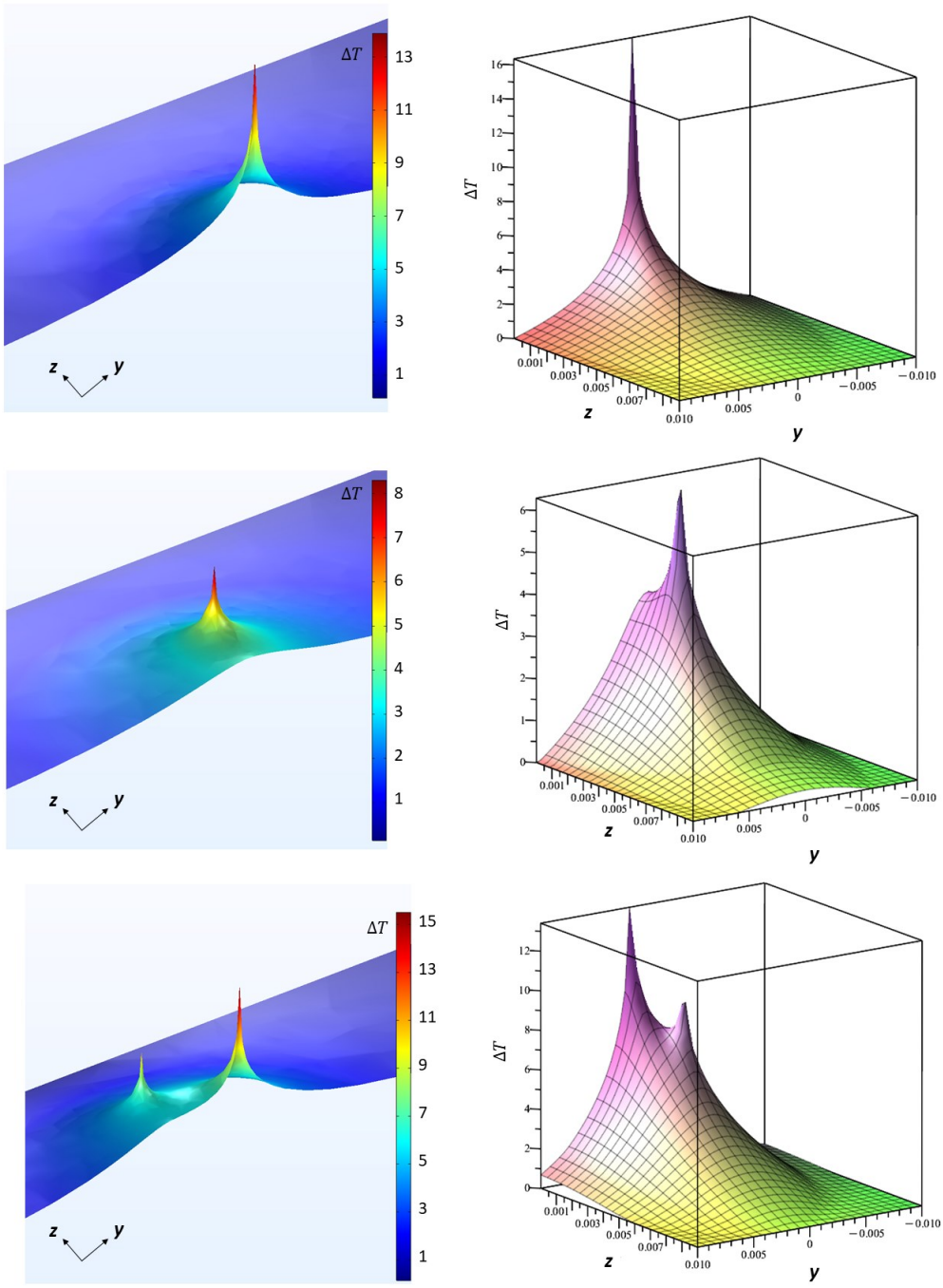


Figure 4.9. Comparison among analytic model and simulations for the reservoir when (a) resistor 1 is self-heated, (b) resistor 3 is self-heated, and (c) resistors 1 and 4 are self-heated.

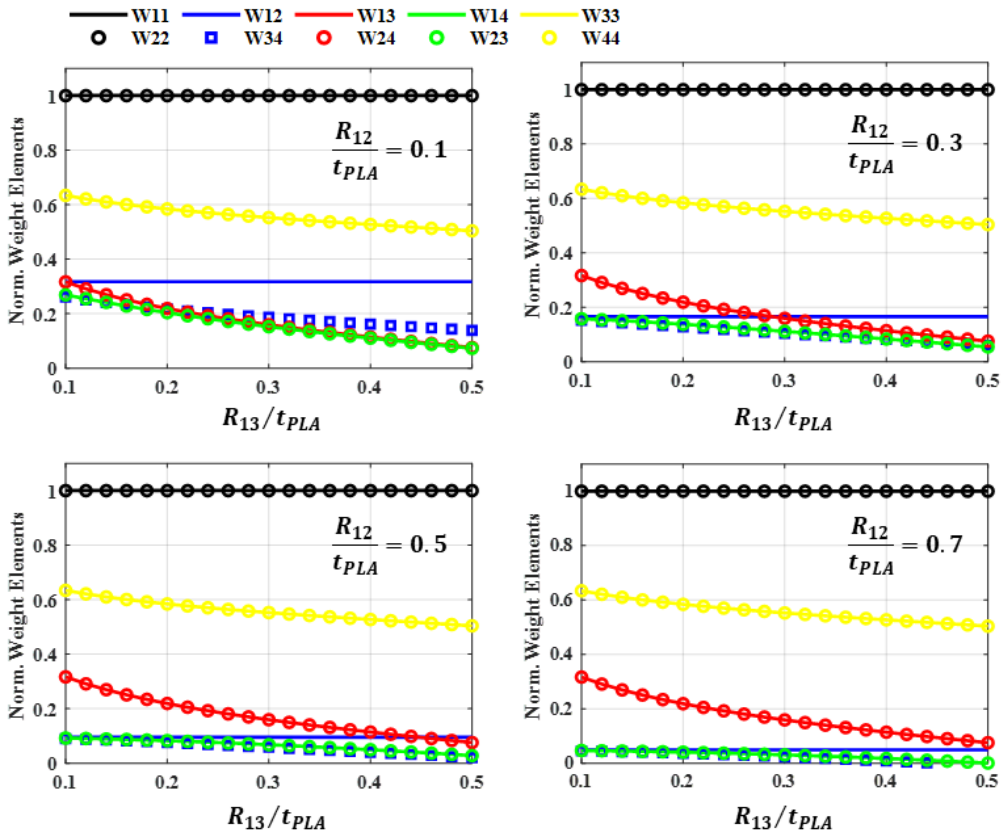


Figure 4.10. Normalized weight elements of the reservoir with four resistors/neurons with different spacings.

4.2.3. Dynamic Response of a Reservoir

It is essential to understand the transient behaviour of the processor because it provides insight into how the system will respond to changes in operating conditions, such as sampling frequency and response time to changes in the injected current (density), and can help to predict the system's performance under various scenarios. In this context, transient heat transfer analysis is used to model, simulate, and analyze the dynamic behaviour of the reservoir of neurons with thermal coupling and to optimize their performance. Most of the heat transients follow the response of the 1st order system through the equation below [101]

$$T(t) = T_f + (T_i - T_f)e^{-t/\tau} \tag{4-15}$$

where, T_i and T_f are the initial and final temperatures of the medium, and *t* represents the continuous time. The thermal time constant, τ, appears in these equations as a parameter

that affects the rate of heat transfer and the rate of temperature change in the system. The time constant depends on the thermal properties and geometry of the materials involved in the heat transfer process.

When the transient response of resistors was studied both in experiments and simulations, two different types of transients were observed. Self-heating resistors exhibit a transient response characterized by two thermal time constants; the internal time constant (τ_{int}) and the substrate thermal time constant (τ_{th}). The internal time constant represents the duration it takes for the heat generated inside the resistors to be conducted away from its center. The substrate time constant signifies how fast the heat is conducted away from the resistors' contact surface deep into the substrate. The thermal time constants rely on the physical properties of the thermistor and substrate, including their dimensions, thermal mass, and thermal conductivity. In a reservoir with effectively large dimensions for the substrate compared to the resistors, the substrate time constant would dominate the response time and determine the transient behaviour of the neurons and their design. Therefore, understanding the factors that affect τ_{th} are important.

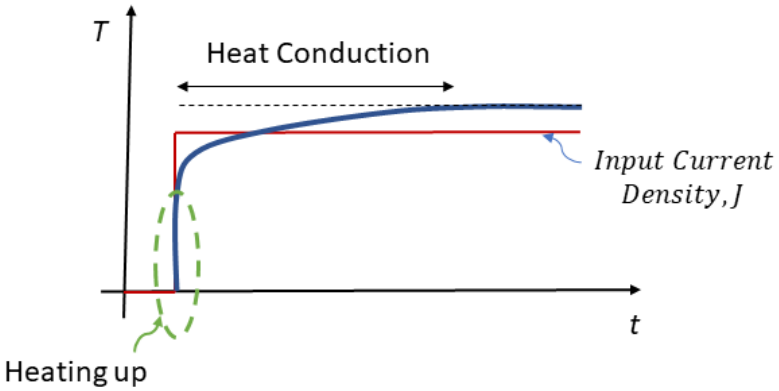


Figure 4.11. Dynamic behaviour of self-heated resistors and the neighbouring ones without self-heating.

Here, let us delve into the analytical understanding of τ_{th}, which plays a crucial role in the transient behavior of the neurons. The regular PLA is assumed as a medium that conducts the generated heat until it reaches equilibrium. By introducing thermal diffusivity as $\alpha = \frac{\kappa_{PLA}}{\rho_m C_p}$, and assuming the temperature does not change significantly in all directions except for r, Eq. (4-4) reduces to:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T(r,t)}{\partial r}\right)=\frac{1}{\alpha}\frac{\partial T(r,t)}{\partial t} \tag{4-16}$$

To solve this equation, we assume that the temperature can be expressed as a product of radial and temporal components as $T(r,t)=R(r)\,.\,T(t)$. By substituting this assumed form into the heat conduction equation in (4-16), we get:

$$\frac{\frac{dR(r)}{dr}+r\,\frac{d^2R(r)}{dr^2}}{r}\,T(t)=\frac{R(r)}{\alpha}\frac{dT(t)}{dt} \tag{4-17}$$

Dividing both sides by $R(r)\,.\,T(t)$, we have:

$$\frac{\frac{dR(r)}{dr}+r\,\frac{d^2R(r)}{dr^2}}{r\,R(r)}=\frac{\frac{dT(t)}{dt}}{\alpha\,T(t)} \tag{4-18}$$

Since the left side depends only on r and the right side depends only on t, both sides must be equal to a constant. Let us denote this constant as $-\lambda^2$ and summarize the equations as:

$$\frac{\frac{dR(r)}{dr}+r\frac{d^2R(r)}{dr^2}}{r\,R(r)}=-\lambda^2 \xrightarrow{r^2\,R(r)} r^2\frac{d^2R(r)}{dr^2}+r\frac{dR(r)}{dr}+\lambda^2r^2R(r)=0$$

$$\frac{\frac{dT(t)}{dt}}{\alpha\,T(t)}=-\lambda^2 \qquad \rightarrow \qquad \frac{dT(t)}{dt}+\alpha\lambda^2T(t)=0 \tag{4-19}$$

We now have two separate ordinary differential equations. The first equation is Bessel's equation, which can be solved using Bessel functions. The second equation is a simple first-order linear ordinary differential equation. The solution considering the initial boundary condition of $T(0)=0$ is given by

$$T(t)=C_1\left(1-e^{-\alpha\lambda^2t}\right) \tag{4-20}$$

where C_1 is an amplitude coefficient. The thermal time constant from equation Eq. (4-20) can be derived as $\tau=\,{}^{1}\!/_{\alpha\lambda^2}$.

For the first equation, the solution depends on the specific boundary conditions or geometry of the problem. For the problem under study, where the heat is generated in the center of the cylinder and conducts away through the regular PLA, the solution would take the form of

$$R(r) = B_1 BesselJ(0, \lambda r) \tag{4-21}$$

where $BesselJ(0, \lambda r)$ is the Bessel function of the first kind of order zero. This function appears as a solution to the cylindrical heat conduction equations.

To determine the value of λ, we need to apply appropriate boundary conditions and solve the problem numerically. A numerical value is found from [101],

$$\tau_{th} = \frac{{L_{ch}}^2}{2.4^2 \alpha} \tag{4-22}$$

where L_{ch} is the characteristic length of the material surrounding the resistors/neurons. In this case, it is the distance from the resistor to the PLA surface with constant temperature. For the resistors in the top layer, the characteristic length is equal to t_{PLA}. This yields τ_h value as 225 seconds for a PLA with a thickness of 10mm encapsulating the resistors. Consequently, a reservoir comprising multiple resistors/neurons will exhibit dynamics with a time constant of approximately 225 seconds, with slight variations based on which resistors/neurons are self-heating. Resistors in the first layer generally possess smaller thermal time constants than those in the PLA's depth. This distinction arises from the fact that the characteristic length for resistors/neurons in the depth of the PLA is modified to $L_{ch} = \sqrt{(t_{PLA} - R_d)^2 + (t_{PLA} + R_d)^2} = \sqrt{{t_{PLA}}^2 + {R_d}^2}$ rather than simply t_{PLA}. Look at the symmetric model in Figure 4.7 for this distance explanation.

Reservoir dynamics when all resistors are self-heated is illustrated in Figure 4.12. To analyze transient responses, simulation data are processed using MATLAB®, employing a combination of two exponentials. This computational approach facilitates the automated calculation of thermal time constants. It is worth noting that resistors within the same layer tend to exhibit similar time constants, but there can be noticeable differences in dynamics between neurons located in distinct layers. Moreover, a deviation from the calculated values for the thermal time constants was observed. This deviation may be

attributed to the fact that, although the dynamics of an individual neuron or resistor are defined by Eq. (4-15), the entire reservoir, comprising multiple interconnected elements, exhibits a unique and distinct dynamical behavior. When all the resistors are self-heated, the dynamic response of the reservoir is described by (assuming a linear medium):

$$\Delta T(t) = \sum_{j=1}^{4} T_{f,j} + \left(T_{i,j} - T_{f,j}\right) e^{-t/\tau_j} \tag{4-23}$$

This equation accounts for the combined thermal effects across the different layers and their respective time constants τ_j. Therefore, the dynamic temperature of the resistors evolves from an initial condition of $\Delta T_i(y, z, 0)$, as expressed as:

$$\begin{bmatrix} \Delta T_1 \\ \Delta T_2 \\ \Delta T_3 \\ \Delta T_4 \end{bmatrix} = \begin{bmatrix} \Delta T_{i,1} e^{-t/\tau_1} \\ \Delta T_{i,2} e^{-t/\tau_2} \\ \Delta T_{i,3} e^{-t/\tau_3} \\ \Delta T_{i,4} e^{-t/\tau_4} \end{bmatrix} + W_{res}(t)^{(4\times4)} . \begin{bmatrix} J_1^2 \\ J_2^2 \\ J_3^2 \\ J_4^2 \end{bmatrix} \tag{4-24}$$

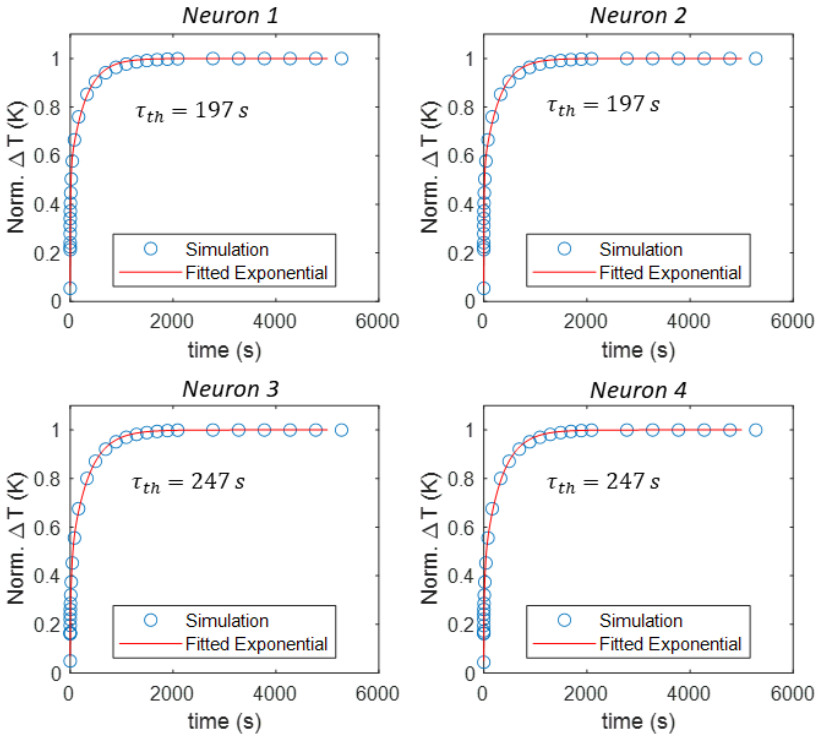


Figure 4.12. Dynamics of the reservoir under different conditions when all resistors are self-heated together.

where

$$W_{res}(t) = \begin{bmatrix} W_{11}\left(1-e^{-t/\tau_1}\right) & W_{12}\left(1-e^{-t/\tau_2}\right) & W_{13}\left(1-e^{-t/\tau_3}\right) & W_{14}\left(1-e^{-t/\tau_4}\right) \\ W_{21}\left(1-e^{-t/\tau_1}\right) & W_{22}\left(1-e^{-t/\tau_2}\right) & W_{23}\left(1-e^{-t/\tau_3}\right) & W_{24}\left(1-e^{-t/\tau_4}\right) \\ W_{31}\left(1-e^{-t/\tau_1}\right) & W_{32}\left(1-e^{-t/\tau_2}\right) & W_{33}\left(1-e^{-t/\tau_3}\right) & W_{34}\left(1-e^{-t/\tau_4}\right) \\ W_{41}\left(1-e^{-t/\tau_1}\right) & W_{42}\left(1-e^{-t/\tau_2}\right) & W_{43}\left(1-e^{-t/\tau_3}\right) & W_{44}\left(1-e^{-t/\tau_4}\right) \end{bmatrix} \quad (4\text{-}25)$$

Each τ_j has a unique value for different neurons. Importantly, as time (t) approaches infinity, $W_{res}(t)$ converges to the static weight matrix derived in the previous section, mathematically represented as:

$$\lim_{t\to\infty} W_{res}(t) = W_{res} \quad (4\text{-}26)$$

Despite the reservoir's continuous operation, it usually interacts with a digital system whose output is sampled at specific time intervals, Δt. This leads to the weight matrix expressed as:

$$W_{res,\Delta t} = \begin{bmatrix} W_{11}\left(1-e^{-\Delta t/\tau_1}\right) & W_{12}\left(1-e^{-\Delta t/\tau_2}\right) & W_{13}\left(1-e^{-\Delta t/\tau_3}\right) & W_{14}\left(1-e^{-\Delta t/\tau_4}\right) \\ W_{21}\left(1-e^{-\Delta t/\tau_1}\right) & W_{22}\left(1-e^{-\Delta t/\tau_2}\right) & W_{23}\left(1-e^{-\Delta t/\tau_3}\right) & W_{24}\left(1-e^{-\Delta t/\tau_4}\right) \\ W_{31}\left(1-e^{-\Delta t/\tau_1}\right) & W_{32}\left(1-e^{-\Delta t/\tau_2}\right) & W_{33}\left(1-e^{-\Delta t/\tau_3}\right) & W_{34}\left(1-e^{-\Delta t/\tau_4}\right) \\ W_{41}\left(1-e^{-\Delta t/\tau_1}\right) & W_{42}\left(1-e^{-\Delta t/\tau_2}\right) & W_{43}\left(1-e^{-\Delta t/\tau_3}\right) & W_{44}\left(1-e^{-\Delta t/\tau_4}\right) \end{bmatrix} \quad (4\text{-}27)$$

After each timestep Δt. When Δt is significantly smaller than the thermal time constants ($\Delta t \ll \tau_j$), the weight matrix approaches zero, indicating that the input at the current timestep has little influence on the thermal response at the current timestep. Consequently, the reservoir retains the previous response, albeit slightly attenuated, and inter-neuron interactions become negligible. Conversely, when Δt is much larger than the thermal time constants ($\Delta t \gg \tau_{ij}$), the input at the current timestep has a substantial impact on the system's response. Interactions among neurons reach their maximum potential value. Thus, for a physical reservoir, the choice of timesteps at which it is sampled, or the sampling frequency ($f_s = \frac{1}{\Delta t}$), plays a crucial role in determining its

performance. The experimental validation of these principles was conducted in the previous chapter along with the next chapter.

Figure 4.13 provides insight into the relationship between thermal time constants and the thickness of the PLA medium for resistors 1 and 3, considering the scenario where all resistors within the reservoir undergo self-heating. The graph illustrates how varying the thickness of the medium influences the system's thermal behaviour. Decreasing the thickness of the PLA medium reduces the thermal time constants, resulting in faster dynamics within the reservoir. Conversely, increasing the medium's thickness corresponds to a slower response. Therefore, the thickness of the medium plays a significant role in shaping the overall temporal characteristics of the reservoir's thermal responses.

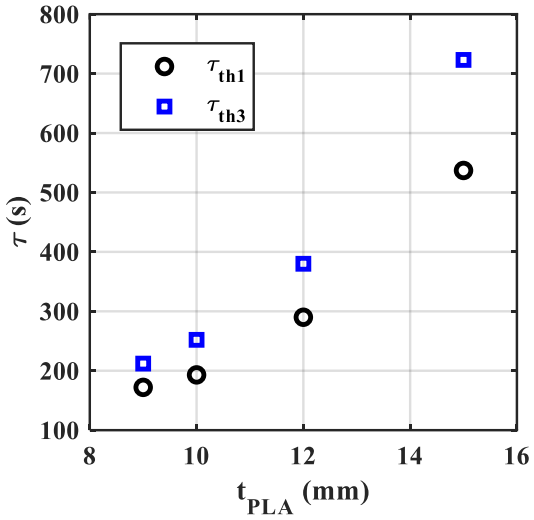


Figure 4.13. Thermal time constants of the resistors 1 and 3 vs. different PLA thicknesses.

4.2.4. Connecting the Analytical Model to Reality

In practice, the resistors we are dealing with are 3D printed and possess rectangular cross-sections, deviating from the cylindrical cross-section assumed in the developed model. To bridge this gap between theoretical assumptions and real-world conditions, a slight adjustment is made to the existing model to accommodate the actual dimensions of the resistors. Specifically, let's consider a scenario where all resistors share the same dimensions: thicknesses of $2r_0$, and widths, $\frac{\pi r_0}{2}$. By setting $r_{1,2} = \sqrt{2}\, r_0$ and $r_{3,4} = r_0$, the previously established model can be effectively extended to predict the

temperature distribution within the reservoir accurately. This modification is depicted in Figure 4.14, illustrating how the adjustment aligns with the rectangular cross-sections of the resistors. Incorporating these adjusted dimensions, simulation results for the rectangular cross-section resistors are presented in Figure 4.15. utilizing the realistic dimensions detailed in Table 4-3. It is notable that the derived model effectively captures and predicts the temperature distribution surrounding the resistors, demonstrating a high degree of agreement between the model's predictions and the simulation outcomes. The successful alignment between the derived model and the simulation results bolsters confidence in its applicability for analyzing and understanding temperature distribution in the reservoir.

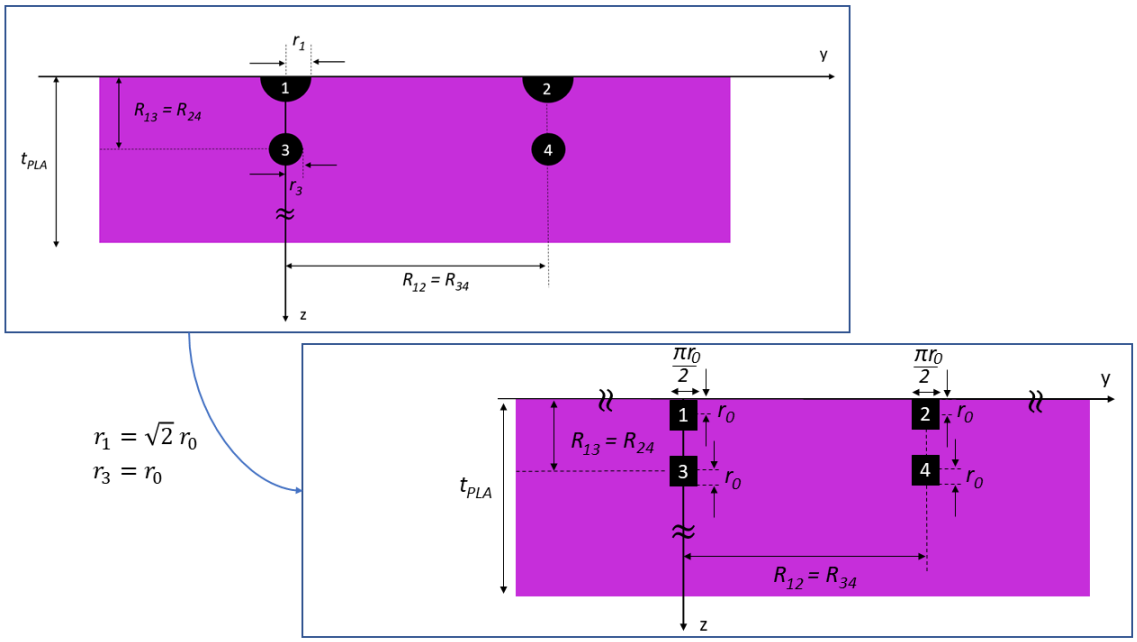


Figure 4.14. Model justifications for the real dimensions: a transisition from resistors/neurons with cylindrical cross-section to rectangular one.

Table 4-3. Real dimensions used for the COMSOL MultiPhysics Simulations.

Parameter	Definition	Value
$2r_0$	Thickness of resistors/neurons	0.5 [mm]
t_{PLA}	Thickness of the 3D-printed piece	5 [mm]
W_{PLA}	Width of the 3D-printed piece	50 [mm]
L_{PLA}	Length of the 3D-printed piece and the resistors/neurons	30 [mm]
R_{12} & R_{34}	Spacing between two neighbouring resistors/neurons in one layer	7 [mm]
R_{13} & R_{24}	Spacing between two neighbouring resistors/neurons in one layer from another	0.25 [mm]

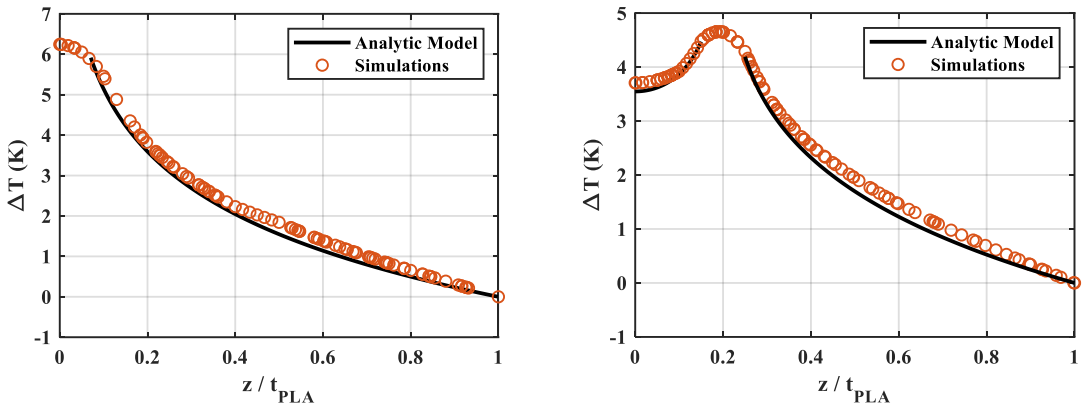


Figure 4.15. Temperature distribution around resistors 1 and 3 when (a) resistor 1 is self-heated and (b) resistor 3 is self-heated for rectangular cross-sections within real dimensions.

To delve further into the behavior of these 3D-printed reservoirs, we explore several key aspects: nonlinear temperature-dependent electrical resistivity, the temperature dependence of thermal conductivity, direction dependence of thermal conductivity, and developing a general reservoir model. These elements will provide a comprehensive understanding of how these resistors interact with temperature variations and how these properties can be incorporated into a broader reservoir modelling framework.

Nonlinear Temperature-Dependent Electrical Resistivity: The electrical conductivity of certain materials exhibits a temperature dependence. Conductive PLA demonstrates a nonlinear relationship between its electrical resistivity and temperature. Despite this characteristic, the presented analytical model does not account for this nonlinear temperature dependence. Nevertheless, incorporating this aspect into the model would introduce implications for various system components. An immediate consequence would be the alteration of the heat generation rate equation within the resistor, where the expression $-\kappa_{PLA}\frac{dT_i(r)}{dr} = q_i$ is impacted. Specifically, the heat generation term $q_i = \rho(T_i)\, {J_i}^2\, r_i$ needs to be adjusted to consider the temperature-dependent resistivity, $\rho(T_i)$. This modification in the heat generation rate would reverberate throughout the model, directly influencing the temperature distribution equations for the resistors/neurons. Furthermore, the weight matrix described in Equation (4-14) would also be affected due to the changes in the temperature distribution equations.

While incorporating such complexity into the model could potentially capture more nuanced behaviours, it's worth acknowledging the challenges that arise. The introduction of nonlinear temperature-dependent resistivity would render the resulting equations challenging to solve. Developing closed-form solutions may become significantly more complex, if possible at all, potentially negating the benefits of adding this level of detail.

Temperature Dependence of Thermal Conductivity: The thermal conductivity of a material, in general, varies with temperature. However, this variation is mild for many materials in the range of practical interest and can be disregarded. In such cases, we can use an average value for the thermal conductivity and treat it as a constant. This is also common practice for other temperature-dependent properties such as density and specific heat. Sufficiently accurate results can be obtained using a constant thermal conductivity value at an average temperature.

Direction Dependence of Thermal Conductivity: The analytical modelling discussed earlier assumes isotropic engineering materials, where properties remain consistent in all directions. This simplification holds true for many practical scenarios, obviating the need to account for directional property variations. However, it is crucial to acknowledge that certain materials, particularly anisotropic ones like fibrous or composite materials, exhibit distinct properties along different axes. Notably, the layer-by-layer 3D printing process introduces the potential for directional discrepancies in thermal conductivity. In this context, the thermal conductivity along the x and y directions might diverge from the z direction. This disparity arises due to factors such as the type and percentage of carbon filler utilized, printing conditions, and the orientation of printed layers. Therefore, while the analytical model discussed earlier does not consider the direction dependence of thermal conductivity, it is paramount to recognize that in practical applications involving anisotropic materials and layer-by-layer 3D printing, variations in thermal conductivity along different axes may impact the system's thermal behaviour.

Develop a General Reservoir Model: A systematic approach involving superposition can be employed to create a comprehensive model accommodating N resistors/neurons within a reservoir. This method enables the derivation of a generalized model capable of addressing diverse configurations and scenarios. Specifically, this model considers a range of neurons within the reservoir, each contributing to the overall temperature distribution. For resistors/neurons situated in the top layer, the temperature distribution

can be expressed as follows: A general model for *N* number of resistors/neurons in a reservoir can be derived by applying superposition on all the *N* neurons knowing that the temperature distribution for the resistors/neurons in the top layer will be (for $r_i < \sqrt{\left(y - R_{l,i}\right)^2 + z^2} < t_{PLA}$):

$$\Delta T_i(y_i, z) = \frac{\rho J_i^2 r_i^2}{k_{PLA}} Ln\left(\frac{t_{PLA}}{\sqrt{\left(y - R_{l,i}\right)^2 + z^2}}\right) \tag{4-28}$$

In this equation, $R_{l,i}$ signifies the lateral distance of the neuron from the coordinate system's origin, which is the neuron located at the center. The provided equation accounts for the temperature distribution as a result of the neuron i within the top layer of the PLA. For resistors/neurons located deeper within the PLA, at a distance $R_{d,i}$ from the top surface, the temperature distribution is determined by (for $r_i < \sqrt{\left(y - R_{l,i}\right)^2 + \left(z - R_{d,i}\right)^2} < t_{PLA}$):

$$\Delta T_i(y, z) = \frac{\rho J_i^2 r_i^2}{k_{PLA}} Ln\left(\frac{{t_{PLA}}^2 - {R_{d,i}}^2}{\sqrt{\left(\left(y - R_{l,i}\right)^2 + \left(z - R_{d,i}\right)^2\right)\left(\left(y - R_{l,i}\right)^2 + \left(z + R_{d,i}\right)^2\right)}}\right) \tag{4-29}$$

These definitions are depicted in Figure 4.16 for a sample reservoir configuration. Their spatial arrangement and thermal interactions then determine the temperature distribution. By establishing this comprehensive framework, practitioners gain a powerful tool to analyze and predict temperature dynamics within a reservoir housing multiple resistors/neurons with different spacings and number of layers. This model's adaptability accommodates different configurations and assists in optimizing system performance for a wide array of real-world applications.

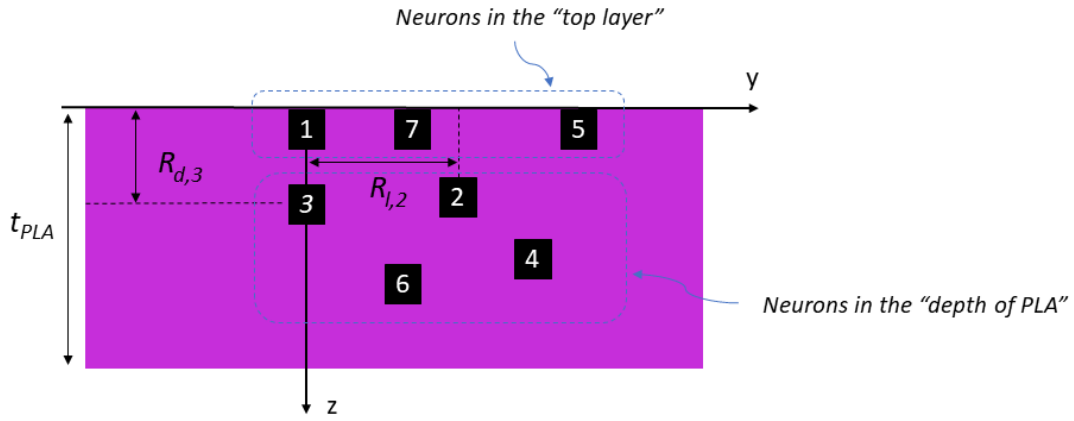


Figure 4.16. A sample reservoir with a general, random configuration defining the neurons in the top layer and in-depth of PLA.

4.3. Computer-Aided Design and Model Verification

In this section, we present a critical component of our research where we bridge the theoretical aspects of our developed model with practical insights through computer-aided design (CAD) and verification. The implementation of a MATLAB® script serves as a valuable tool to not only create physical reservoirs but also to validate our theoretical framework. A simplified flowchart outlining the pivotal steps involved in this process is depicted in Figure 4.17. The MATLAB® script is designed to receive critical parameters that are fundamental to creating and developing physical reservoirs. These parameters are: the number of neurons, specific neuron locations (if provided; otherwise, it generates a reservoir with a random spatial distribution), neuron dimensions, and PLA dimensions. Using these inputs, the script configures the reservoir. Leveraging the insights from our general model, the script employs superposition to create isothermal contours. These contours serve as powerful visualizations, providing an intuitive representation of the temperature distribution within the reservoir. Furthermore, the script goes beyond visualization; it generates both static and dynamic weight matrices based on a specified input signal. The generation of static and dynamic weight matrices enables an assessment of how the reservoir processes information over time. This dynamic evaluation allows us to gain insights into how the system's dynamics evolve in response to varying inputs, providing an understanding of its computational behavior. To validate our theoretical model and ensure its practical utility, the MATLAB® script conducts a critical verification step. It compares the temperature across each resistor or neuron within the physical reservoir to the corresponding temperatures obtained from COMSOL Multiphysics

simulations. This verification process plays a pivotal role in solidifying the theoretical foundation of our model and confirming its real-world applicability. Figure 4.18 provides a visual illustration of the verification step, offering a clear depiction of how the temperature verification is conducted and emphasizing the alignment between our analytical model and the empirical data obtained through simulations.

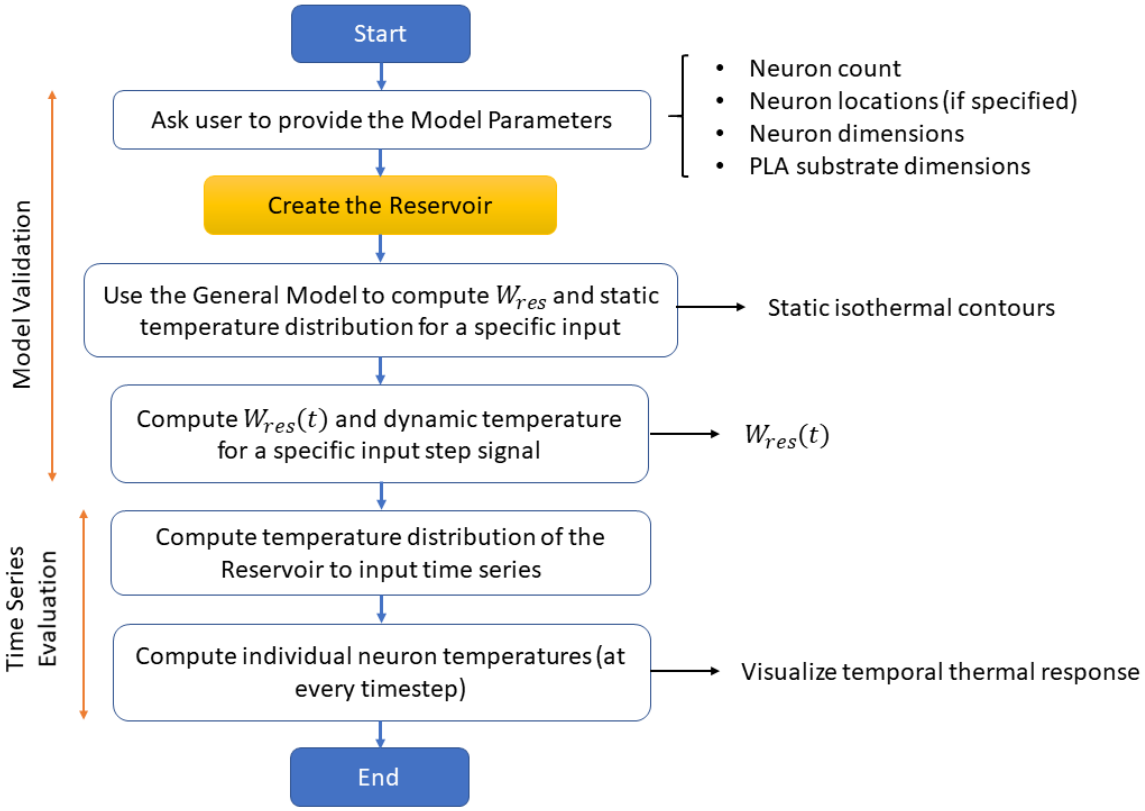


Figure 4.17. Flowchart of the processes involved with creating a physical reservoir. It outlines the script initialization, parameter handling, visualization, and dynamic evaluation with a time series input.

In the subsequent phase, we introduced a time series as an input to the model and simulations, unlocking the ability to observe the dynamic response of the reservoir system in real time. Within the MATLAB® script, the temperature distribution across the entire reservoir was calculated and monitored the temperature of individual neurons at one-second intervals. This approach allowed us to closely track and analyze the behavior of the system over time. As illustrated in Figure 4.19, a high level of agreement between the simulation results was observed and the predictions generated by our theoretical model when subjected to the provided time series input. This agreement highlights the model's ability to faithfully capture the intricate dynamics of the reservoir computing system in real-

time scenarios. It validates not only the static aspects of our model but also its dynamic predictive power, further affirming its practical applicability and relevance in real-world RC applications.

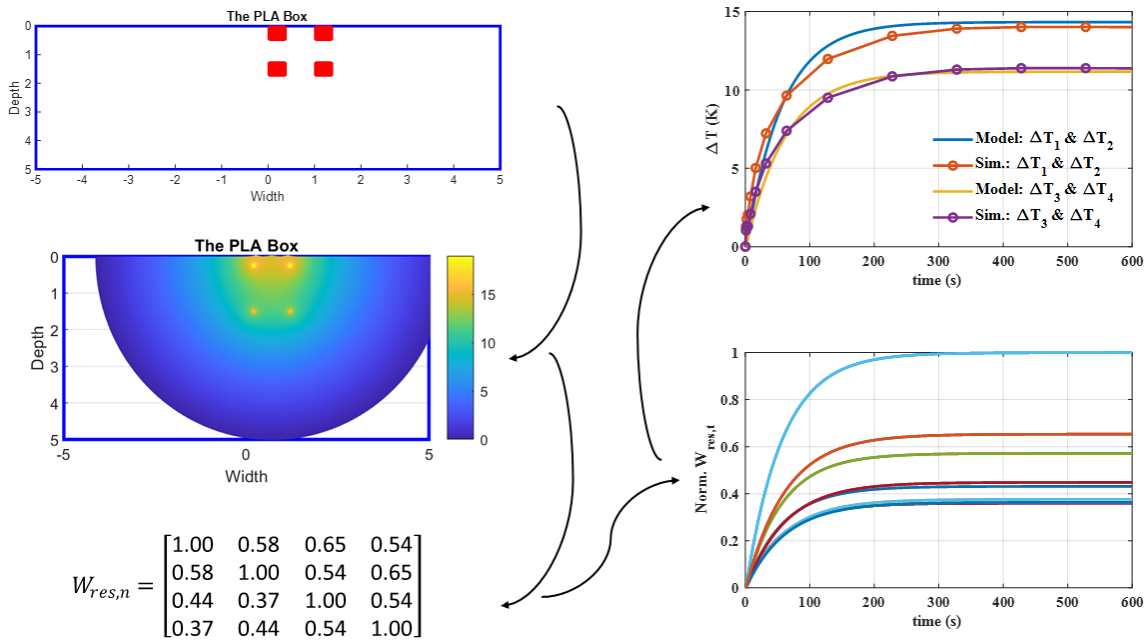


Figure 4.18. (a) Step-by-step outcomes of the MATLAB® script following the developed general model in the previous section. The reservoir is configured first, isothermal contours are visualized, both static and dynamic weight matrices are derived based on a specified input, and a comparison between the temperature across each resistor/neuron within the physical reservoir to the corresponding temperatures obtained from COMSOL Multiphysics simulations are made.

Using the previously outlined reservoir, we successfully addressed a nonlinear problem with memory constraints. Figure 4.20 illustrates the input signal introduced into the reservoir alongside NARMA1 designated as the output for prediction by the reservoir. This comprehensive performance assessment encompasses both the reservoir constructed within the MATLAB® environment, utilizing the developed analytical model and the reservoir simulated using COMSOL Multiphysics to solve the nonlinear task. Remarkably, the performance of both reservoirs demonstrates a substantial level of comparability.

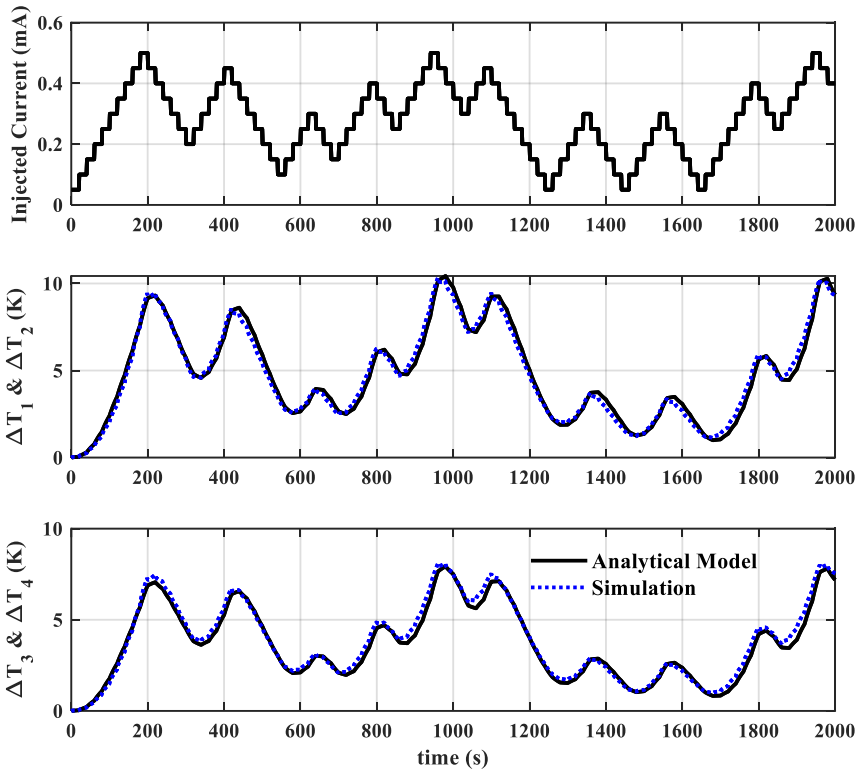


Figure 4.19. Comparison of the developed general model and the simulations from COMSOL Multiphysics in response to a time series signal in the input.

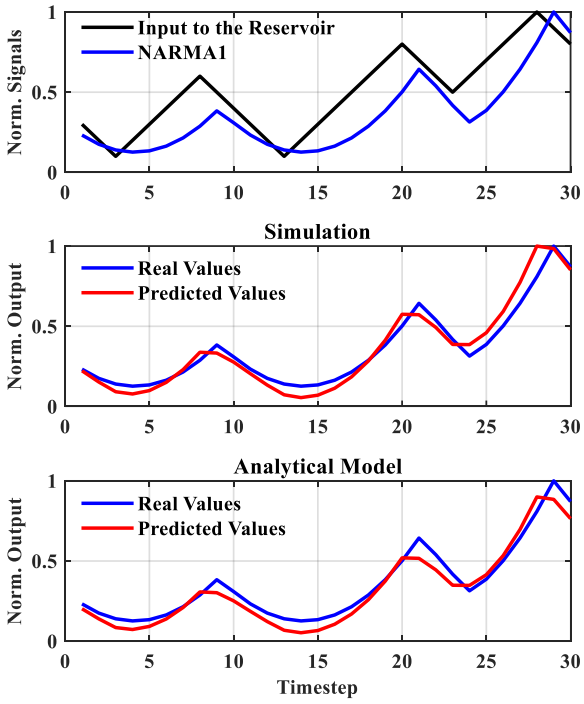


Figure 4.20. Performance of the reservoir built in MATLAB® environment using the developed analytical model with the reservoir simulated in COMSOL Multiphysics in solving a nonlinear problem.

4.4. Summary

In this chapter, we elaborated on the idea of 3D-printed processors as a solution to address the requirements for computing units in processing applications. Energy transfer could be a valuable source of information transfer in hardware reservoirs and identified two types of energy that can be harnessed to this end in the 3D printed processors working based on the self-heating as a source of memory and nonlinearity within the reservoir. We conducted analytical studies on the reservoir's thermal energy transfer and weights matrix to explore this concept further.

This chapter ends with laying a synergy between physical RC and software RC. In the realm of physical RC, the selection of appropriate physical components or materials is a crucial step. These components could encompass a range of possibilities, including mechanical, optical, or electronic elements. The choice of components is not arbitrary; it depends on the desired dynamics, compatibility with the overall system design, and how the chosen materials respond to different stimuli. These choices determine the nonlinearity the physical RC exploits and dictate how the reservoir's behaviour changes over time. Different systems with exponentially decaying behaviour, various forms of damping, and even chaotic systems can offer the desired dynamic properties that show fading memory. For instance, mechanical oscillators exhibit exponential decay, where the energy or amplitude of oscillations gradually decreases over time due to damping mechanisms. Another example can be found in resistor-capacitor circuits, where discharging a capacitor through a resistor results in an exponential decrease in voltage across the capacitor over time ($\sim e^{-t/\tau}$), where τ is the time constant of the system.

Moving on, we discuss how energy is exchanged when we consider the interaction between physical components within the reservoir. These mechanisms can be of various types—electrical, mechanical, thermal, or hybrid. The selection of these coupling mechanisms profoundly impacts how energy flows, signals propagate, and computations are carried out within the physical RC system. Balancing the connectivity patterns and choosing appropriate coupling mechanisms is pivotal for achieving the desired computational performance. The physical layout and geometry of the reservoir also play an equally crucial role. This includes how the components are arranged and distributed spatially and the system's overall structure. These factors determine how strongly components are interconnected and the system's overall complexity. These aspects

collectively determine a key parameter called the "spectral radius" of the physical reservoir, which characterizes the system's dynamic behaviour.

Physical RC systems require suitable input signals to operate and generate desired outputs in the context of input signals. These input signals can take various forms, such as electrical, optical, thermal, etc., depending on the nature of the physical components being used. The generated output signals correspond to changes in physical quantities like displacements, currents, or light intensity. Ensuring that input and output signals are designed efficiently and controllably is critical for achieving the desired computational performance. The concept of input scaling in physical RC corresponds to adjusting the amplitude of input signals to guide the reservoir to its intended operating state. This ensures that the input values fall within a suitable range that the reservoir can process effectively. It's analogous to tuning the gain or amplification of input signals in electronic systems.

The analogy between physical and software RC parameters, which is summarized in Table 4-4, provides a bridge that enables researchers and engineers to translate their understanding and strategies from one domain to the other, fostering a richer experience and more effective design strategies. This synergetic relationship amplifies our comprehension and design capabilities, paving the way for more informed and efficient development across both fields. As we transition to the next chapter, we delve into the practical realization of these concepts, showcasing the proposed 3D printed computing platform's ability to process real-time data and its remarkable classification capabilities with real sensory information. Through tangible demonstrations and benchmark assessments, we underscore the platform's potential to perform intricate computations, ultimately ushering in a new era of versatile and powerful computing solutions.

Table 4-4. Synergy between Software and Physical RC.

Parameters	Software RC	Physical RC
Activation function	$f(.)$	Material/device nonlinearity e.g., $\rho(T)$
Leaking rate	α	$e^{-1/f_s\tau}$ for an exponentially decaying system
Input scaling	a	Gain/scaling of the input signal
Spectral radius	ρ	Coupling strength
Neuron connections	Information flow	Coupling/energy flow
Reservoir size	# neurons	# devices/electrodes

Chapter 5.

The Proposed 3D-Printed Computing Platform

This chapter discusses the ability of the proposed 3D printed computing platform to process real-time data and its classification capability for real sensory data. We will demonstrate that a simple structure printed with regular 3D printers can be driven and used with common measurement tools to perform sophisticated contextual computations, including standard benchmarks and a demonstration of user activity detection from sensor data. This chapter partly contains my article published in *Advanced Intelligent Systems entitled "*A 3D-Printed Computer*"* [102]. Correlations between memory capacity, nonlinearity, and sampling rates with this computer were examined. Despite its simplicity, the computer can tackle complex standard tests and is used to solve the practical problem of user activity detection. Adding to the computational capability of the demonstrated computer is simply achievable by printing additional computational nodes. At a material cost of less than $1, this processor can be used next to existing intelligent systems for contextual signal processing. It can also be embedded within the structure of 3D-printed intelligent systems, enabling the realization of cognizant 3D-printed systems.

5.1. The 3D-Printed Reservoir

One of the challenges in implementing physical RC is the choice of a physical system to be used as the reservoir. In the previous chapter, we showed that 3D-printed resistors could be a suitable candidate for developing RC computers based on ESN topology. The nonlinearity and time dependence in resistors' responses make them ideal for use as elements in a reservoir. In addition, the coupling between the elements can be achieved through electrical or thermal coupling, as introduced earlier. Here, the proposed 3D-printed reservoir using 3D-printing technology is described. Several resistors were printed close to each other such that the heat generated by one would reach and affect its nearby neurons. Thus, the reservoir structure can be interpreted as a three-layer reservoir in which the weighted connections between the neurons are realized through thermal and random electrical couplings. Moreover, the neurons on each layer are electrically coupled to each other. Three layers of resistors printed of Carbon-PLA composite were stacked on each other with a gap in between, filled with pure PLA as the insulating material, as shown

in Figure 5.1. The lateral dimensions of the 3D printed reservoir are 3.9 cm × 3.5 cm, while its height depends on the thickness of the resistors (i.e., the number of printed layers). The width of the resistors is 1.6mm, and the gap between the resistors (layers) is set to 0.2mm. Three samples with similar lateral dimensions that differed only in the thickness of the conductive layers (resistors) were 3D printed. The thicknesses of the resistors in Samples 1 to 3 were 2 mm, 1 mm and 0.5 mm, respectively.

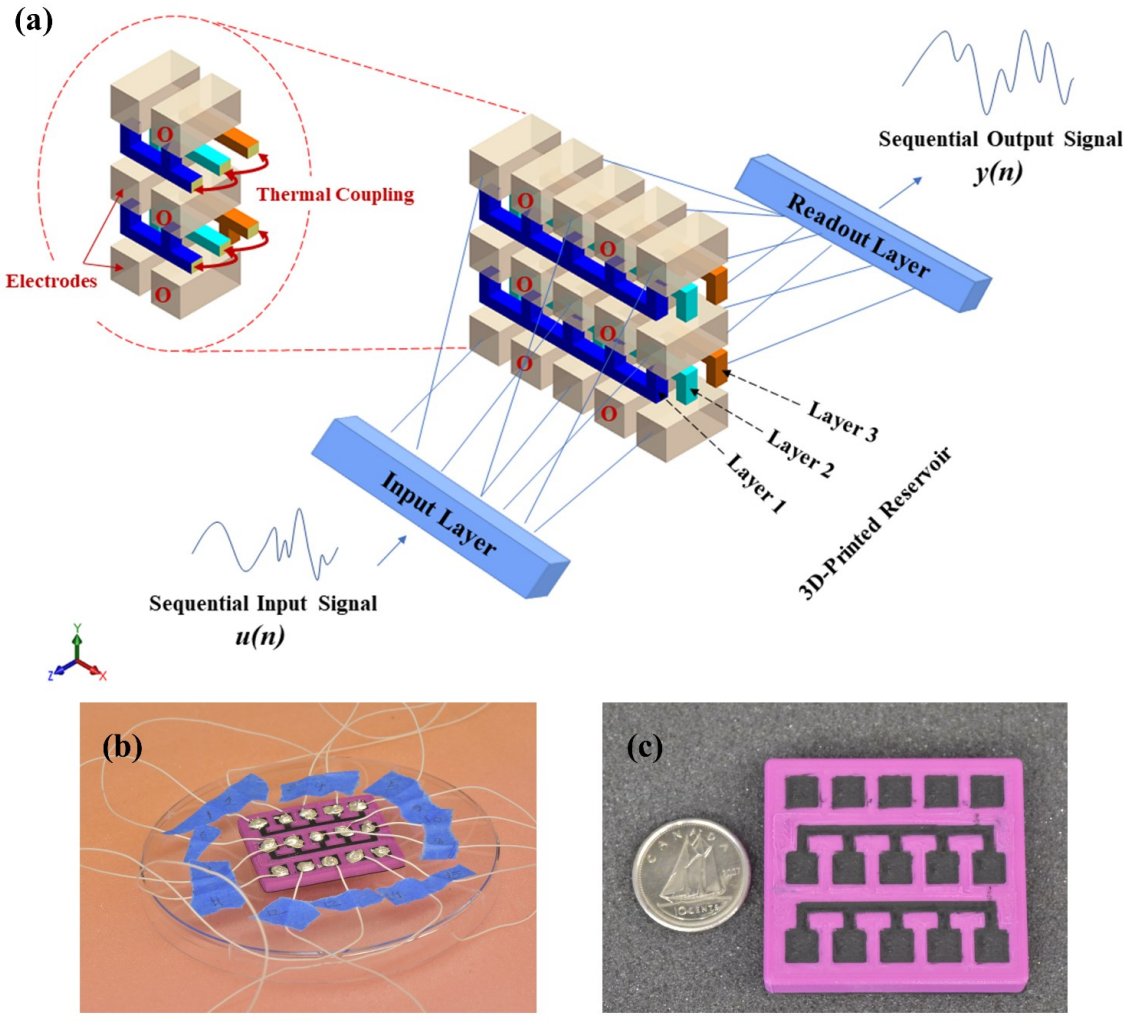


Figure 5.1. Components of the 3D printed computer. (a) The proposed reservoir structure; (b) The 3D-printed reservoir with electrical connections for applying or reading electrical signals; (c) the top view of the computer with three conductive layers printed between the insulating material next to a Canadian ¢10 coin.

These resistors' nonlinear, time-dependent responses and their coupling satisfy the requirements for building an ESN. Contact pads were used to apply the input signals or to read the data from specific nodes. The input layer is driven by an analogue shift register which shifts and scales the input signal, $u(n)$, multiple times (i.e., past three samples $u(n-1)$, $u(n-2)$, and $u(n-3)$) and maps the generated signals to the reservoir. The output layer is trained by running a linear regression on all the outputs from the reservoir in MATLAB®.

These structures were initially studied to evaluate the nonlinearities and time dependencies of the responses of neurons at different locations and the couplings between them. Figure 5.2 shows test results from these samples on individual neurons' responses to electrical excitations. The hysteresis in the resistor's response, shown in Figure 5.2 (b), represents both memory and nonlinearity. Hysteresis refers to a phenomenon in which the output of a system depends not only on its current input but also on its past inputs. Hysteresis in a system or device can then be considered a form of memory. Together, these two characteristics can make it difficult to predict the behaviour of a system. It was also observed that the neurons' responses might be affected by injecting a control current, I_C, into different neurons in the structure (Figure 5.2 (c)). This effect may be used to bias different neurons post-fabrication to make them respond differently from other similarly fabricated devices, adding another parameter that can be used to enhance the reservoir complexity.

Figure 5.2 (e) demonstrates a resistor's measured electrical and thermal responses on the top layer of the 3D printed processor over time due to a step current input. After the current injection, the voltage across the resistor and its temperature slowly rise until thermal equilibrium with the environment is reached. In this case, the resistor exhibited a thermal time constant (TTC) of ~62s. TTC will differ for different neurons in the structure and depend on the thermal boundary conditions, such as being embedded within the structure or being exposed to the environment on one or more surfaces. This property, too, adds a degree of randomness and helps with using the 3D-printed structure as a contextual processor.

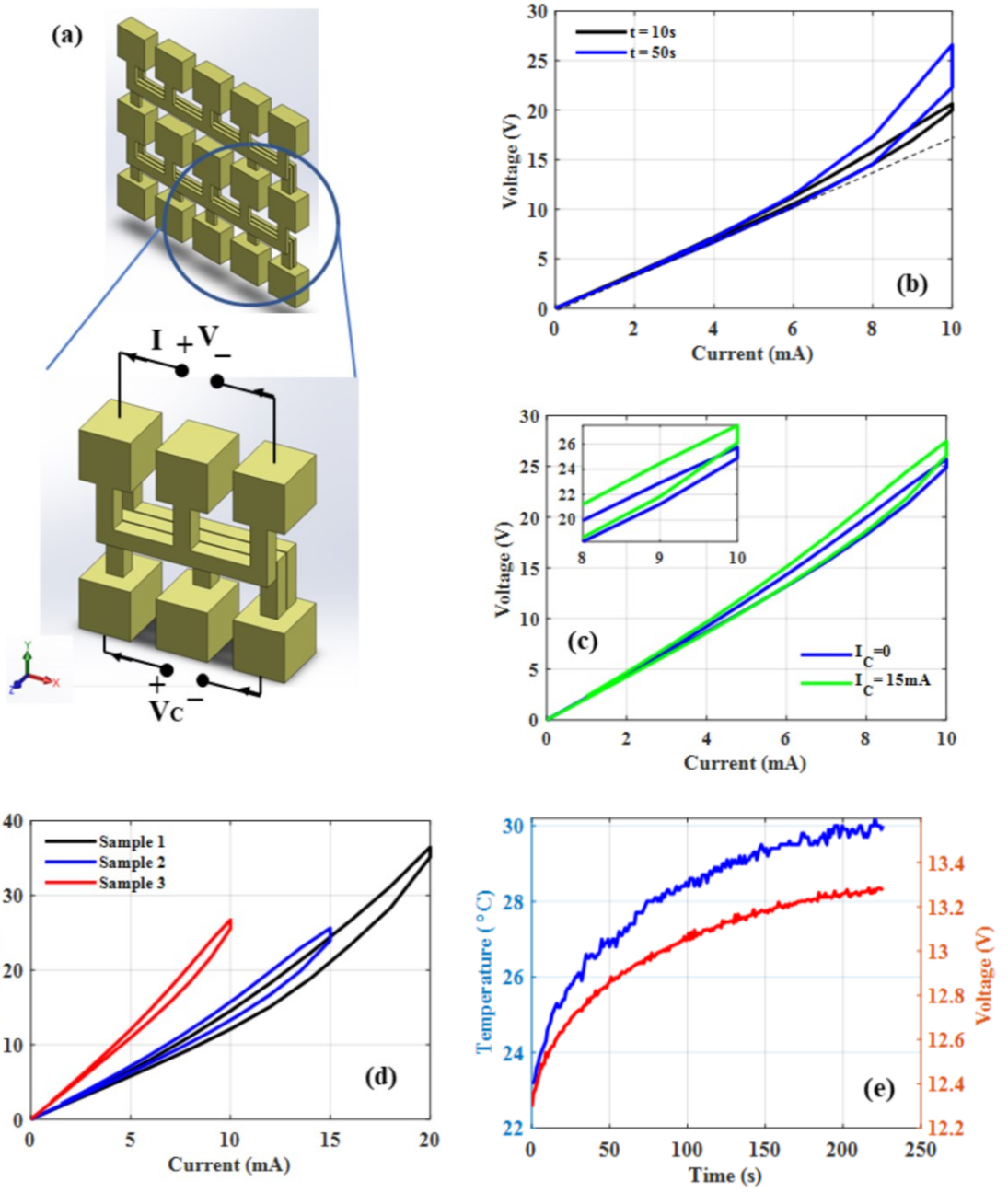


Figure 5.2. **Electrical response of printed structures. (a) Schematic of the conductive paths in the fabricated reservoir and its close-up view; (b) The nonlinear I-V response of the reservoir obtained by sweeping a current through two one of the resistors in steps with a time delay of 10s or 50s between the steps; (c) The effect of control current, I_C, on the response of a resistor; (d) Comparison of the I-V responses of three reservoir samples with similar lateral dimensions but different thicknesses for the conductive resistors (e) Temperature and the voltage across the two electrodes over time in response to an I=8mA step current input.**

In physical reservoir computing, reservoir richness refers to the ability of a physical system to generate a high-dimensional and nonlinear response to a given input signal, which is a key property required for performing complex computations such as pattern recognition, time-series prediction, and control. It is a critical factor in the design and optimization of physical reservoir computing systems, as it determines the computational power and accuracy of the system. The physical system's complexity and nonlinearity determine a physical reservoir's richness. It can be characterized by various measures, such as the system's dimensionality (the combination of the number of neurons and their interconnections), the system's sensitivity to initial conditions, and the system's memory capacity, a measure of its ability to store and retrieve information.

One aspect of richness is the nonlinear or linear response of individual neurons in the reservoir. Neurons in a reservoir can exhibit various nonlinear and linear response characteristics. The nonlinear and linear responses of neurons in a reservoir can interact with each other in complex ways, leading to the emergence of even more complex behaviours. For example, the nonlinear responses of some neurons in a reservoir can modulate the linear responses of other neurons, resulting in dynamic and adaptive processing capabilities. The nonlinear and linear interactions of neurons in a reservoir can also create feedback loops that allow the reservoir to store and manipulate information over time. Thus, richness in reservoir computing encompasses a wide range of complex and diverse nonlinear and linear responses of neurons, as well as their interactions, that enable the reservoir to perform complex computations. Figure 5.3 demonstrates some characterization results for Sample 3. The neurons in the printed reservoir exhibited linear and nonlinear responses to different ranges of input signals, as expected.

A single printed computer provides numerous possibilities to arrange input and output layers by applying or reading signals to different contacts. See Table 5-1 for the arrangement used in this study. Figure 5.4 illustrates thermal images of the reservoir at different instances of time after the application of an input to the reservoir, showing the evolution of reservoir characteristics through heating and its subsequent effect on the IV characteristics of the neurons.

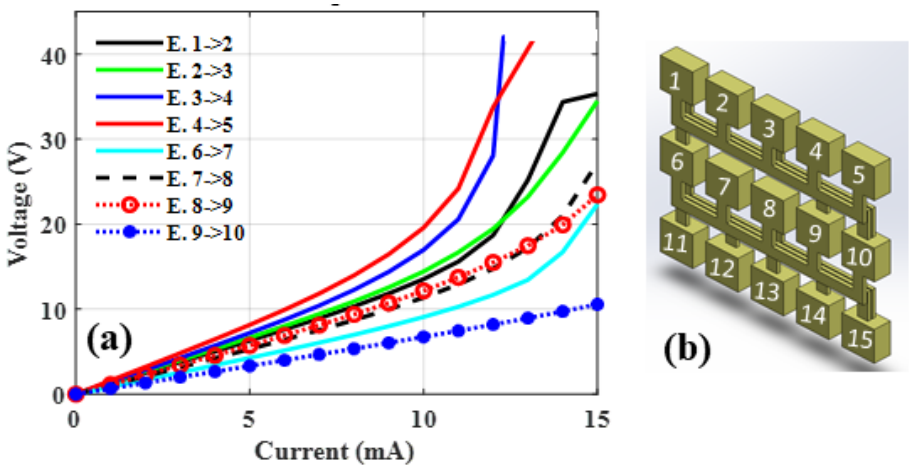


Figure 5.3. **Electrical and thermal response of coupled components. (a) IV characteristics of various pairs of electrodes; (b) The electrode labels.**

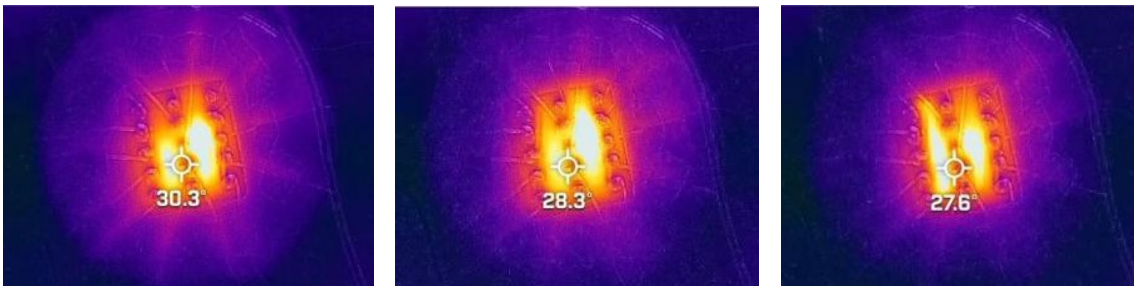


Figure 5.4. **Thermal images of the reservoir at different computation times.**

Table 5-1. **Arrangement of the electrodes in the reservoir and their functions.**

Electrode #	Status
2, 4, 7, 9, 12, 14	Output States (W_{out})
3, 8, 11	GND
1, 6, 10, 13	Input States (W_{in})
5, 15	V_C

Figure 5.5 illustrates how the reservoir computing (RC) approach uses the reservoir outputs to distinguish between different events in a time-series signal. The incoming signal is first fed into the reservoir, and time-shifted copies of the signal are applied. The reservoir outputs, which represent the nonlinear and linear combinations of the input signals, are then combined using weights that were previously calculated during the training stage. This combination process results in a filtered and transformed version of the original signal, which can be used to distinguish between different events.

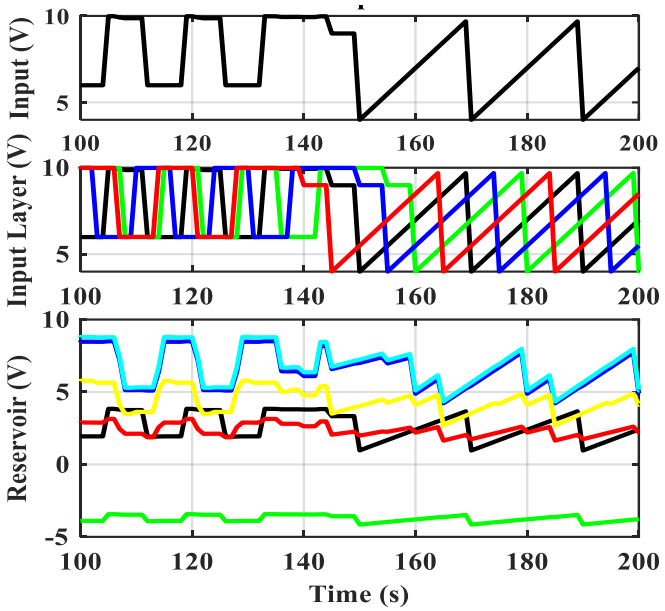


Figure 5.5. A sample time-series signal (top), its time-shifted copies applied to the reservoir input by the analogue shift-register (middle), and the reservoir output (bottom).

The reservoir's output neurons are a mapping of the reservoir states to the desired output of the system. How the output neurons respond to different initial input signals provides insight into the complexity of the reservoir. For example, if the reservoir is linear, then the output neurons may respond in a highly correlated way to an input. In contrast, in a nonlinear complex reservoir, the output neurons may react more independently. This property depends on the range of the input signals (i.e., operating point) and the number of nonlinear neurons within the reservoir. Four identical and two different control signals (as control signals) were introduced to the reservoir without any time shifts. The output neurons were read when the reservoir was initiated at a specific input value and increased with an increment of 1V. Figure 5.6 illustrates the different reservoir states that each of the output neurons (shown in coloured markers) were initiated and evolved. All the output neurons respond in a highly nonlinear way to certain initial input signals, especially when they are self-heated enough (i.e., $\geq 3V$). Sometimes, the states were distributed and the distribution did not change but a shift happened (b). Sometimes, the states were initially close to each other but departed and diverged in the next step (c). Sometimes, two furthest states can get closer and two far states can get away from each other (c)-(e). These suggest that the reservoir is highly complex and nonlinear and can respond complexly or unpredictably. However, another important factor in the complexity of a system is the time-dependence.

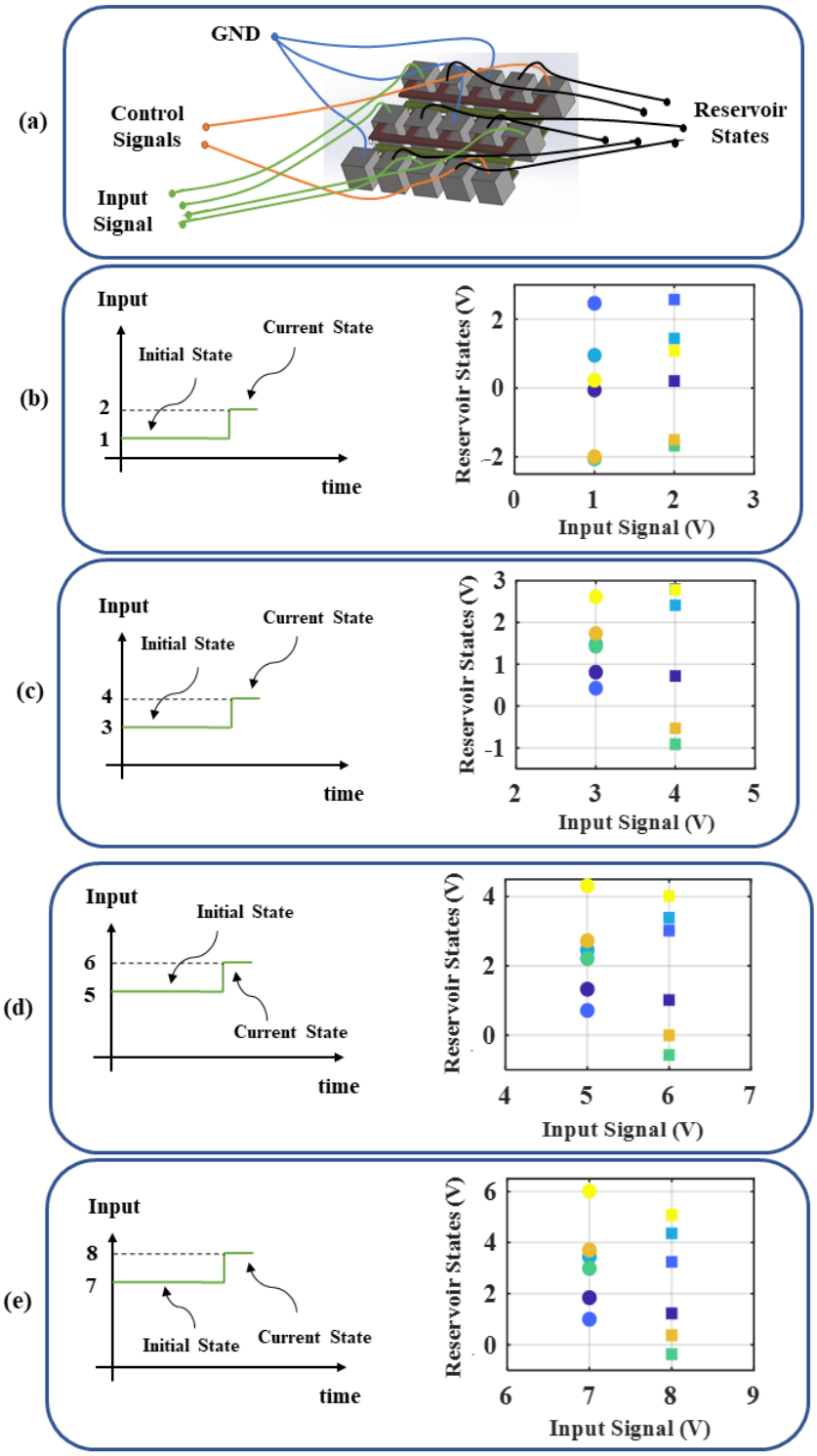


Figure 5.6. **Initiation and evolution of the reservoir when stimulated with different input signals. Responses of the output neurons are illustrated to capture the evolution. (a) The 3D-printed reservoir with specified input, output and ground electrodes. (b) A shift in the responses happened, (c) departing and diverging happened, (c)-(e) two furthest states got closer and two far states got away from each other.**

5.2. Performance Evaluation based on Standard Tasks

According to what we concluded from the previous section, we generated a sample dataset which includes 360 samples in the range of [3 10] V. Two hundred fifty samples from this data (~70%) were randomly selected and used to train the output layer weights. The remainder of the data set was used to validate the computer performance. Figure 5.7 shows the response and the analysis of the response of the 3D printed processor to NARMA tasks of varying order ($n = 1 \dots 10$). This simple processor with only 18 nonlinear neurons performs well for $n \leq 7$. The performance of the processor can be improved through several simple approaches, including adding computational neurons, varying the control signal, or time-multiplexing the tasks between parallel computers.

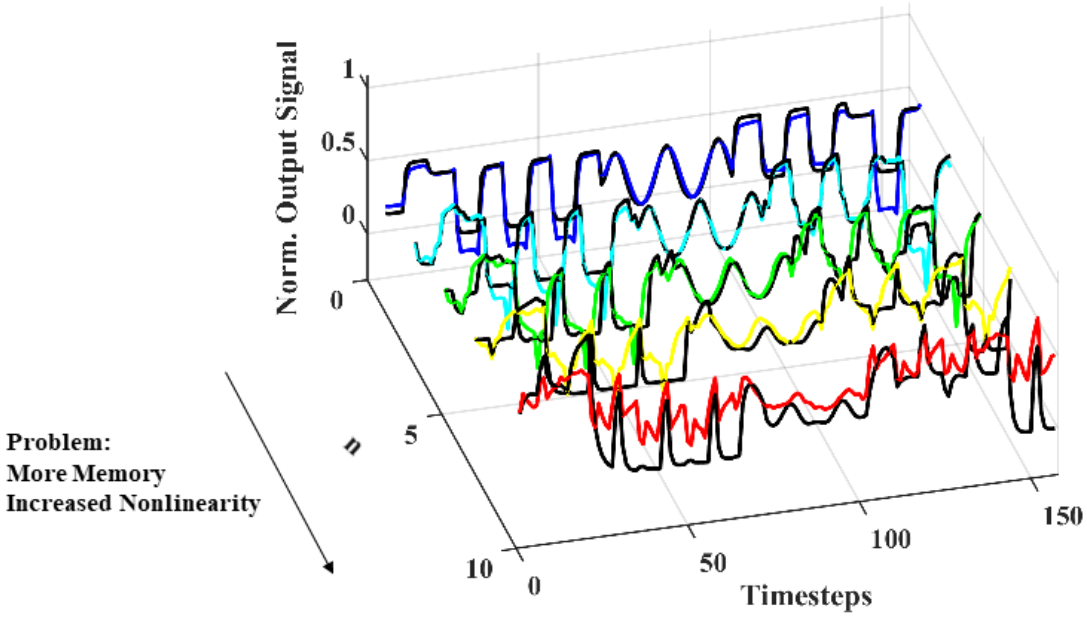


Figure 5.7. Assessment of the computer performance to solve computational tasks with varying orders of nonlinearity and memory. (a) The reservoir response to predicting a time-series input produced by varying orders of NARMA task with $f_S=f_{S3}$.

It is well-known that different dynamical systems have different characteristic timescales (and/or frequencies) that govern their behaviour. Therefore, to excite different dynamics in a given system, it is necessary to apply input signals at appropriate sampling rates (and/or frequencies) that are tailored to the specific system. For instance, in chaotic systems, minor variations in the input signal or sampling rate can result in vastly different

trajectories and outcomes, highlighting the sensitivity of these systems to their initial conditions. On the other hand, in more regular systems, such as periodic or quasi-periodic oscillators, the input signal can be designed to match the system's natural frequency, allowing for optimal energy transfer and efficient excitation of the desired dynamics. Therefore, the importance of selecting the appropriate input scaling and sampling rate cannot be overstated when dealing with dynamic systems. It is crucial to carefully tailor these parameters to the unique characteristics of each system to drive it to the desired dynamic area.

We first study this phenomenon using a NARMA task of varying order and then move to a more straightforward memory capacity task. The NARMA test helps study the compromise between the input signal's sampling rate and retainable memory within the system [103]. The system state approaches thermal equilibrium at a slow sampling rate (i.e., low data rate). Although the reservoir exhibits the most nonlinear response in this case, and hence the ability to solve complex problems, it may have lost information about distant past events. On the other hand, a fast sampling rate (i.e., high data rate) reduces the reservoir nonlinearity needed for contextual computing but helps the system retain more information about past events. Figure 5.8 (a) and (b) demonstrate the performance of the reservoir in solving NARMA tasks of varying orders with different sampling rates. As can be seen, both fast and slow sampling rates result in more significant errors. Therefore, proper selection of sampling frequency is essential in achieving optimal performance from the reservoir in terms of accuracy.

Memory Capacity (MC) is a key concept in evaluating the performance of a contextual processor when dealing with temporal data. It is defined as the ability of a processor to retrieve past information from the reservoir using the linear combinations of its internal states. The higher the MC, the better the processor's ability to recall past events, making it useful for a wide range of applications that require processing time-series data. Indeed, a high memory capacity means that the system can retain information about past events for extended periods and use this information to solve complex problems. This can be seen as a measure of how well the system can capture and model the underlying dynamics of the data it is processing (*i.e.,* higher expressivity). Thus, a system with increased memory capacity is considered more expressive, as it can solve more complex problems and capture more intricate patterns in the data.

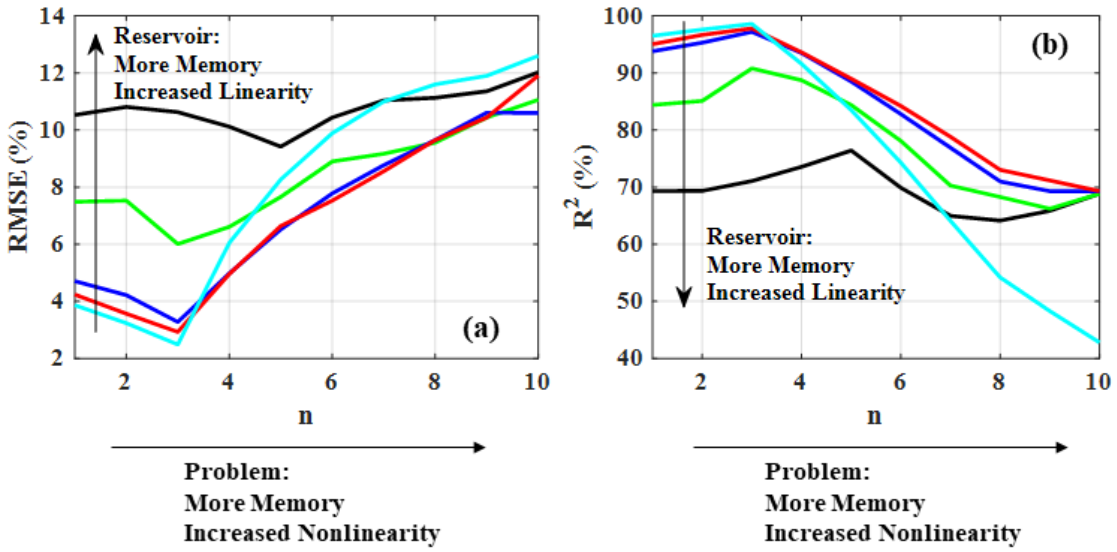


Figure 5.8. Comparison of the reservoir performance when solving NARMA tasks of different orders with different sampling frequencies.

On the other hand, memory capacity can be seen as a measure of how well the system can handle large amounts of temporal data (i.e., scalability). For instance, a system with high memory capacity can take longer data sequences, making it more scalable. However, some other design parameters, such as the size and connectivity of the reservoir, the type of nonlinearity, and the input/output mapping, can also play a significant role in determining the processors's expressivity and scalability.

Figure 5.9 demonstrates how the memory capacity of the reservoir is affected by the sampling frequency when solving a NARMA3 task. As can be seen, a slow sampling rate results in the computer operating with the least error (because of the high nonlinearity) but also a small MC (~2) because the system *forgets* about past events. On the other hand, a high sampling rate results in poor accuracy, but the system remembers many past events (~12). A balance may be struck for this reservoir by choosing a sampling frequency around 6/TTC to achieve good computation accuracy with a high MC (~6). Thus, a slow sampling rate results in the system operating with increased nonlinearity, leading to higher computational accuracy. However, this comes at the cost of a smaller memory capacity, limiting the types of problems that the system can solve.

On the other hand, a high sampling rate can help the system remember many past events, enabling it to solve a broader range of problems. However, this also results in poor accuracy due to reduced nonlinearity. Again, in the context of neural networks,

expressivity refers to the ability of a network to represent complex functions, while scalability refers to the power of a network to handle large datasets or increase complexity efficiently. The sampling rate affects both properties in the context of a physical contextual processor. A slow sampling rate can improve the system's nonlinearity, allowing it to represent more complex functions. This, in turn, leads to higher expressivity. However, it may limit the system's ability to handle larger datasets, reducing scalability. Conversely, a high sampling rate can help the system remember past events, increasing its ability to handle larger datasets and potentially increasing scalability. However, this may come at the cost of reduced nonlinearity, limiting its expressivity.

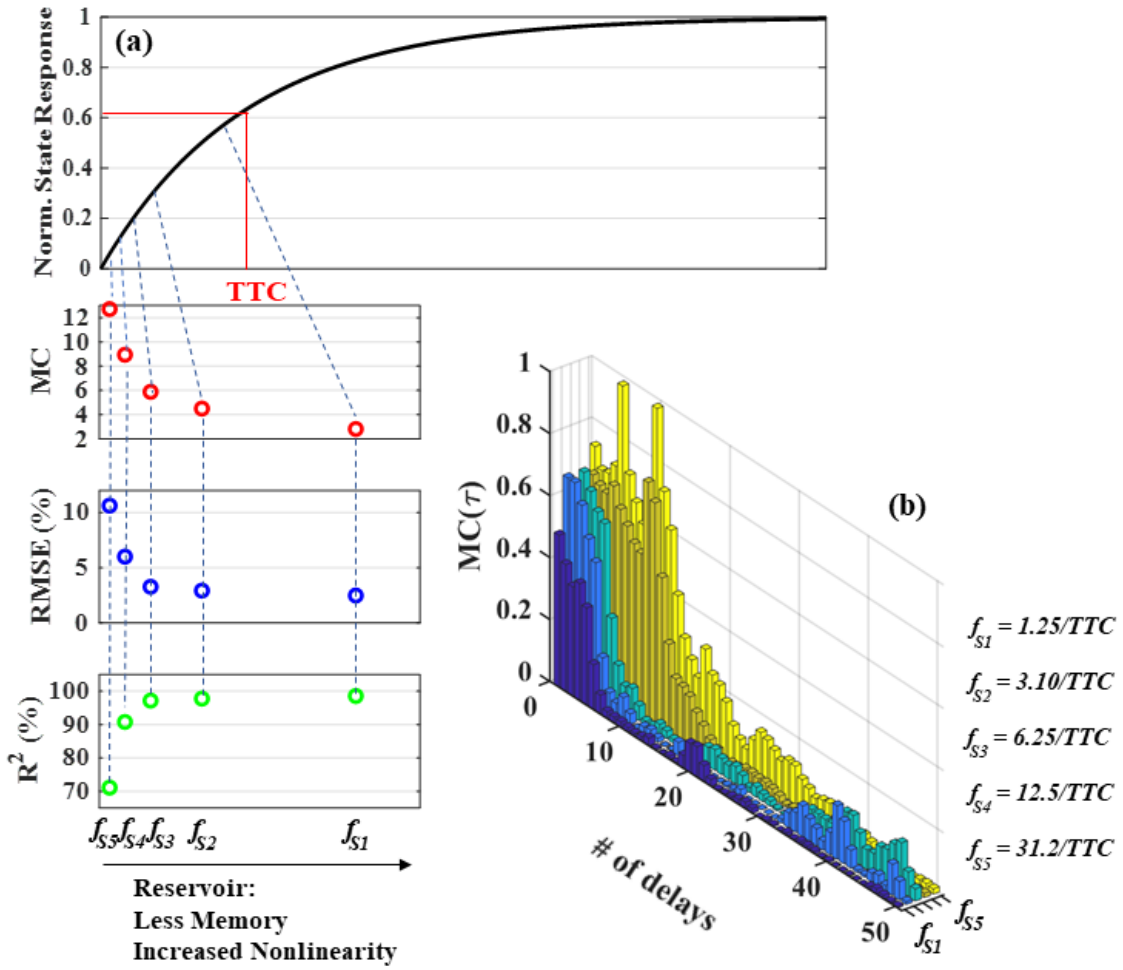


Figure 5.9. Demonstration of the dependence of reservoir memory capacity, RMSE, and R2 to sampling frequency when solving a NARMA5 task. The graph at the top shows a typical step response of one of the reservoir's nonlinear neurons, which is used to estimate TTC and set sampling frequencies.

Therefore, the choice of sampling rate is a crucial factor in determining the expressivity and scalability of a contextual processor. We can establish a trade-off between the two properties to achieve optimal performance depending on the problem. The behaviour of the reservoir can be influenced by adjusting the sampling rate. Consequently, we can alter the system's ability to solve certain problems or retain past information. Therefore, the sampling rate can be an added degree of freedom to the system.

In general, the optimal performance of physical processors depends on the combination of network complexity, sampling rate, and nonlinearity. The processors need to be designed to attain a certain level of performance through their physical design and optimizing dimensions, number of neurons, types of materials, and other parameters that affect their nonlinearity and time responses.

To justify our conclusions about the memory integrated within the reservoir and to prove its presence in the reservoir, we compared the performance of the 3D-printed processor with a feedforward neural network (FNN) constructed in MATLAB®. The FNN, although consisting of multiple layers and neurons, cannot store and retain temporal information, which limits its performance in certain tasks. In contrast, the 3D-printed processor integrates memory within the reservoir, allowing it to consider past inputs when making predictions, resulting in superior performance. The results in Figure 5.10 showed that the 3D-printed processor outperformed the FNN, consisting of 3 hidden layers with 18 neurons. Each neuron in the FNN used a sigmoid activation function. It was found that increasing the number of neurons in the FNN improved its performance to some extent, but it still could not match the performance of the 3D-printed processor.

This discrepancy in performance between the 3D-printed processor and the FNN can be attributed to the added degree of freedom that memory integration provides. The memory integrated within the reservoir adds a layer of complexity to the system, allowing it to process and retain temporal information. The added complexity and degree of freedom enable the 3D-printed processor to perform better than the FNN in specific tasks, demonstrating the importance of memory and temporal processing in computational systems. The comparison between the 3D-printed processor and the FNN can also be related to expressivity and scalability. Due to its ability to integrate memory and process temporal information, the 3D-printed processor can be considered more expressive and

scalable than the FNN, which lacks these capabilities. The added degree of freedom provided by memory integration expands the range of tasks the 3D-printed processor can perform, making it a more versatile and powerful computational tool.

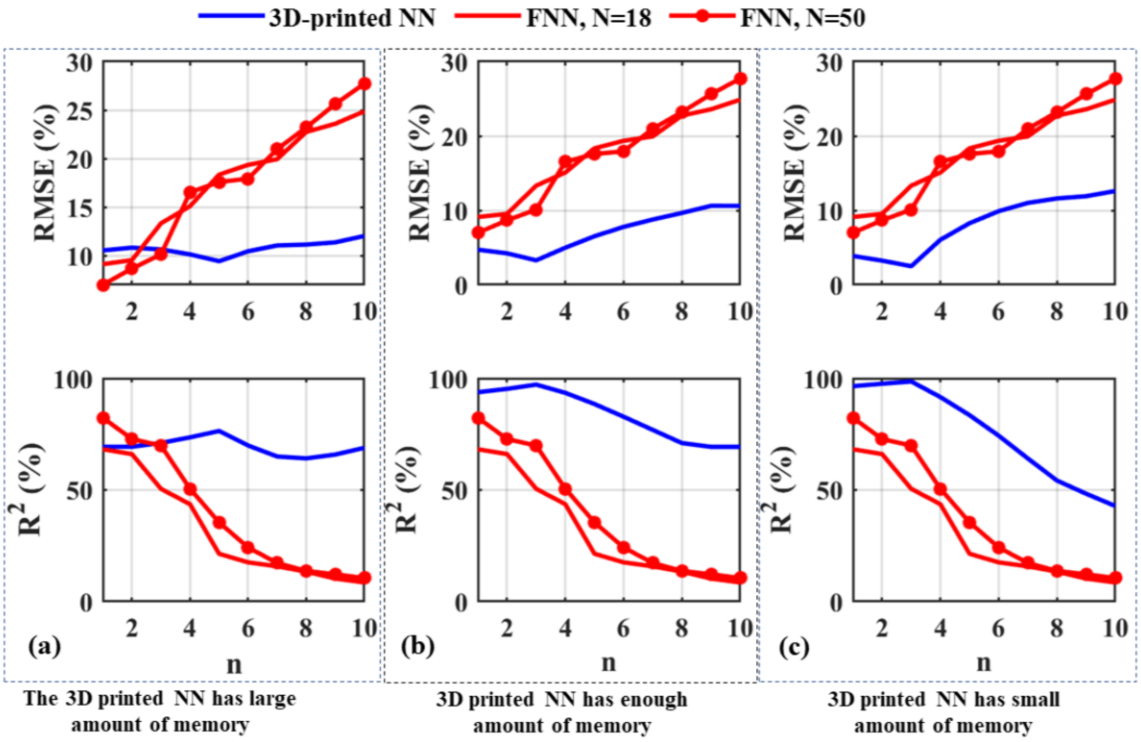


Figure 5.10. The 3D-printed processor's performance (when driven into different dynamical regions by adjusting the sampling rate) compared to a feedforward neural network in solving computational tasks with varying orders of nonlinearity and memory.

5.3. Near-Sensor Data Processing

Activity detection is an important application of wearable systems, which can be used to monitor human conditions and health. In recent years, with the development of mobile Internet-of-Things (IoT) platforms, wearable devices have become more advanced and sophisticated, with various sensors and onboard computational capabilities that communicate with remote servers. One of the most common features of many wearable devices is their ability to detect the type of human activity. This ability is useful for various applications, including sports training, healthcare, and security monitoring.

To develop statistical models for activity detection, data preparation and feature extraction are crucial steps. The quality of the data and the features extracted from it will

significantly impact the accuracy and performance of the statistical models. In this case study, we used data from a SensorTile kit from STMicroelectronics, which includes a pair of microcontrollers, a 3-axis accelerometer, a 3-axis gyroscope, a 3-axis magnetometer, as well as pressure, temperature, and humidity sensors [104]. The accelerometer signals were used to identify the user activity between walking, stationary or on an elevator. The data were collected by attaching a SensorTile kit to a user's wrist and storing the data during different activities.

The inertial signals from all three axes (a_x, a_y, a_z) needed to be included in developing the statistical model. However, it was noted that there is likely a strong correlation between the three features. Therefore, one may make the acceleration data independent of the module orientation by calculating an equivalent signal that retains the important information. This equivalent acceleration was calculated using the formula

$$a_{eq} = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{5-1}$$

This signal was then used to extract features provided to the 3D-printed computer for activity detection. The next step was to define time windows during which we would try to identify the activity. After some trials, a measurement window of 5 seconds was selected in this case. This window should be long enough to let us calculate meaningful features and short enough to allow for the timely detection of activity changes.

During each measurement window, the mean, standard deviation (STD), root mean squared (RMS), and number of peaks of a_{eq} were calculated as features. Since the range of feature values can vary widely, normalizing the features can help prioritize the features by a human or event identification by a machine. These features were then transformed into analog voltage signals and fed into the 3D-printed computer to indicate the user's activity data in each time frame. Figure 5.11 shows a sample of raw data collected from the accelerometers (a_{eq}) as well as features extracted from 5-second timeframes for different activities.

The processor was trained on 196 instances of labelled user activity data. The output layer weights were determined through linear regression, and the processor's performance was evaluated using 60 cases of labelled activities that were not included in

the training dataset. We measured the processor's accuracy in detecting patterns in the data and compared the results with those of typical machine learning algorithms. Figure 5.12 depicts the entire procedure. A truth table shows the performance of the 3D-printed computer in detecting patterns in the data after training. The processor performs the data processing with 93.3% accuracy, 92.5%, 92.8%, and 100% sensitivity in determining whether the user is in an elevator, stationary, or walking. Notice that the sampling period, in this case, is 5 seconds, corresponding to a sampling frequency of about 12/TTC. This level of performance for such a limited set of input features is on par with the performance of typical machine learning algorithms on the same data.

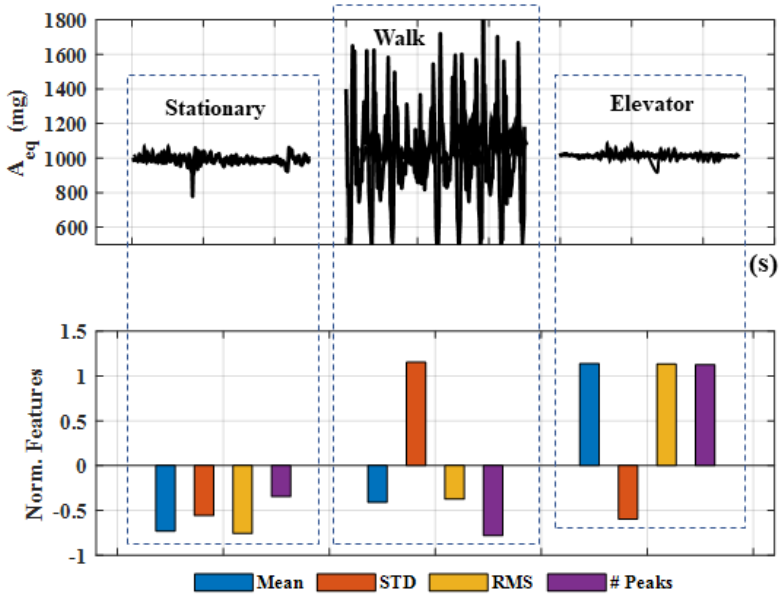


Figure 5.11. Sample raw sensor data and normalized extracted features for different activities.

5.4. Power and Speed Considerations

We used common 3D printer for printing the processor with promising computing capability. The average power consumption for our processor in its current version is approximately 125mW, while the thermal time constant is ~62s. This power is required for heating each neuron to around 15 degrees above room temperature. Our current prototype consumes a high amount of power and has a lower speed in processing data. However, other 3D printing technologies, such as micro stereolithography (micro-SLA), can achieve minimum feature sizes in the range of tens to hundreds of micrometres. These

technologies utilize a light source (such as a UV LED) to cure and solidify photopolymer resin layer by layer into the desired 3D shape. The high precision of these technologies is due to the fine control of the light source and the ability to produce fine layers of resin. The minimum layer thickness in micro-SLA 3D printing technology can range from a few micrometres to tens of micrometres, depending on the specific machine and process used. Some micro-SLA systems can produce features with layer thicknesses as low as 1-2 micrometres. These capabilities are promising in printing processors with lower power consumption and higher speeds.

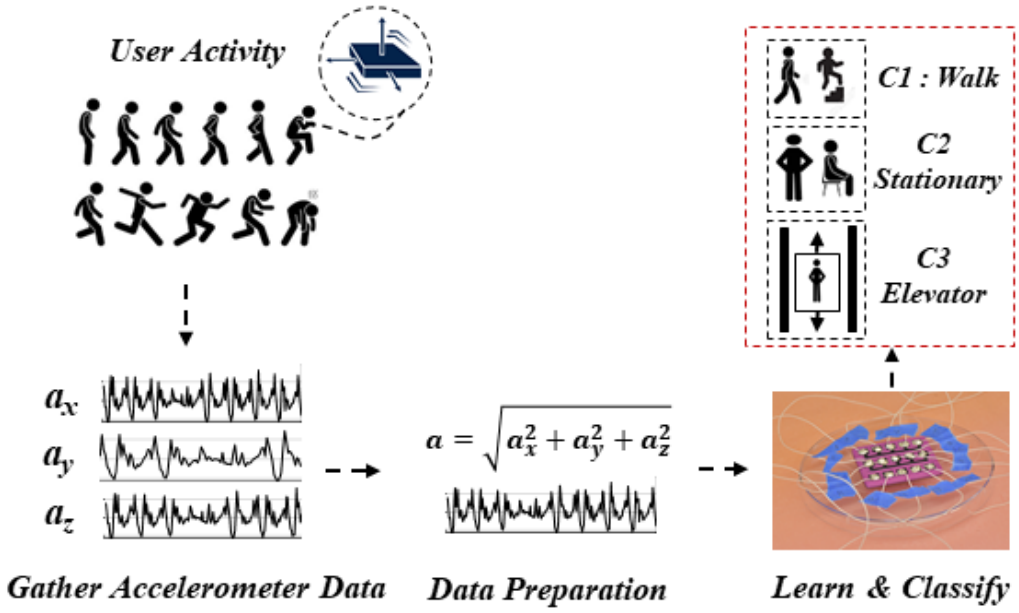


Figure 5.12. Activities detection procedure by the 3D-printed processor. The data were collected using a 3-axis accelerometer on a wearable device worn on a user's wrist during different activities. A truth table shows the performance of the computer in recognizing user activity from sensor data.

The power required for each neuron is given as

$$P_E = RI^2 = \rho_e(T)\left(\frac{I}{A_x}\right)^2 (d_{th}.W.L) \tag{5-2}$$

where d_{th}, W, and L are the thickness, width and length of the resistors, respectively. Therefore, the power consumption would be reduced by three orders of magnitude to 125μW if all the dimensions were reduced by a factor of 10.

On the other hand, the thermal time constant is related to the thermal conductivity, specific heat capacity, and density of the printing material, as well as the thickness of the entire structure, t_{PLA}.

$$\tau = \frac{\rho_m C_p}{\kappa_{PLA}} {t_{PLA}}^2 \quad (5\text{-}3)$$

Therefore, the thermal time constant reduces by two orders of magnitude to 620ms if all the dimensions are reduced by 10. The resulting processor would be easily and feasibly able to process 10 samples/s while consuming an average power of 125μW. The power consumed by a single neuron is 7μW/neuron (*i.e.,* 0.4μW/weight).

Transitioning from a 3D-printed reservoir to silicon offers a remarkable leap, due to smaller feature sizes and better thermal properties reducing thermal time constant by at least 5 orders of magnitude, resulting in a maximum timescale of ~0.6ms and a minimum processing speed of 10000 samples/s. An average power consumption of 7nW/neuron (i.e., 0.4nW/weight) is achievable by dimension reduction from mm range to tens of μm range using readily available microfabrication technologies, which can be translated to 0.4pJ/weight for silicon reservoirs. This shift underscores the pivotal advantages of silicon reservoirs compared to a standard feedforward neural network running on the most efficient supercomputer on the Green500 list consuming 15pJ/weight [105]. All the above is summarized in Table 5-2.

Table 5-2. Power consumption and speed justifications with change in material and the 3D printing technology.

	Resistor dimensions	**Timescale**	**Sampling rate**	**Power consumption**	
3D-Printed Reservoir	0.5mm x 1.6mm x 4.6mm	60s	0.1 samples/second	7mW/neuron	0.4mW/parameter
	50μm x 160μm x 460μm	0.6s (100X)	10 samples/second (100X)	7μW/neuron (1000X)	< 0.4μW/parameter (1000X)
Silicon-based Reservoir	5μm x 16μm x 46μm	600μs (100,000X)	10,000 samples/second (100,000X)	< 7nW/neuron (1,000,000X) *	< 0.4nW/parameter (100,000X) *
* Ignoring the thermal dissipation to the environment.					

5.5. Comparison between the RC and Neural Networks

We developed the processor to predict time series data and tasks involving sequential data. We will conduct a benchmark analysis to compare the proposed 3D printed processor's ability to process data with state-of-the-art software neural networks, including feedforward neural networks (FNNs) and recurrent neural networks (RNNs). The performance of the processor will be evaluated using predictive accuracy, memory capacity, and number of parameters (trainable weights). Predictive accuracy is the most commonly used metric to evaluate the performance of NNs on time series prediction tasks. We will use *RMSE* and R^2 to assess and compare the accuracy of the proposed 3D-printed processor to the FNNs and RNNs. We will measure the ability of the network to accurately predict future values in the NARMA task based on past values. As mentioned earlier in this thesis, MC measures the network's ability to store and retrieve information from previous inputs, which is important for tasks such as sequence generation. We will train the network on a sequence of inputs and evaluate its ability to generate the next element based on the previous elements. In a machine learning model, "parameters" refers to the values the model must learn to make accurate predictions on new data. In a neural network, the parameters typically include the weights and biases associated with the neurons in the network. These parameters are adjusted during the training process to minimize the error between the predicted outputs and the true outputs. The number of parameters in a machine learning model is important because it affects its capacity and ability to learn complex patterns. Models with too few parameters may underfit the data and fail to capture important patterns. At the same time, models with too many parameters may overfit the data and memorize the noise in the training set. Therefore, choosing an appropriate number of parameters for a given problem is important. In addition, the number of parameters can also affect the computational resources required to train and use the model. Models with a larger number of trainable parameters may require more time and memory to train and may be slower to make predictions at inference time. Therefore, balancing the model's capacity with its efficiency and practicality in a given application is essential.

A feedforward network was created as a benchmark analysis to compare the performance of the 3D-printed processor against a more traditional machine learning approach. The feedforward neural network (FNN) is a type of artificial neural network that

is widely used in machine learning applications. In MATLAB, the FNN was created using the *feedforwardnet* function, which constructs a multilayer perceptron (MLP) architecture. The *feedforwardnet* function allows the user to specify the number of hidden layers, neurons in each layer, the activation functions used in each layer, and other parameters. The FNN consisted of an input layer with four neurons, one hidden layer of size 24, and an output layer, as depicted in Figure 5.13.

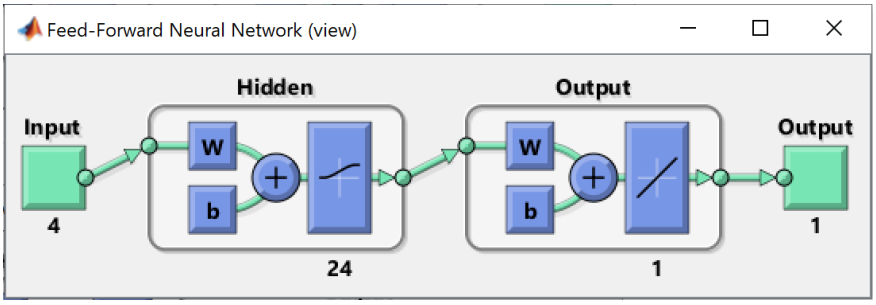


Figure 5.13. The feedforward neural network created to compare with the proposed processor.

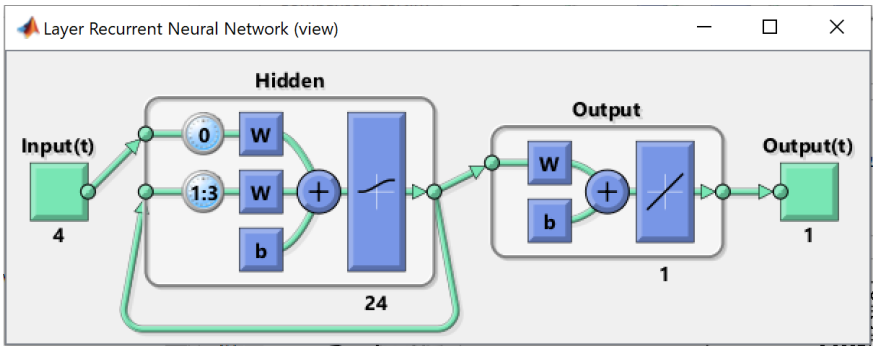


Figure 5.14. The layered recurrent neural network created to compare with the proposed processor.

A recurrent neural network (RNN) was created for performance comparison. The RNN was designed with a fully connected network structure. This memory retention capability made the RNN suitable for processing sequential data, such as time series data, where the current output depends on both the current and previous inputs. The RNN consisted of an input layer with four neurons, one hidden layer of size 24, and an output layer, as depicted in Figure 5.14. A layered RNN, or *layrecnet*, was implemented in MATLAB®. This architecture includes multiple recurrent layers stacked on each other, each taking the previous layer's output as input. In MATLAB®, the *layrecnet* can be easily implemented using the *layrecnet* function, which inputs the number of layers, the number of neurons in each layer, and the delay for each layer.

In comparing the performance of the proposed 3D printed processor with that of RNNs and FNNs in modelling nonlinear and dynamic systems, as summarized in Table 5-3, it was found that RNNs and the proposed processor outperformed FNNs. Figure 5.15 illustrates the time-series processing performance of the proposed processor, FNNs, and RNNs. Because of having a larger number of parameters than FNNs, RNNs could model the system's temporal dynamics and capture long-term dependencies, resulting in higher prediction accuracy. Regarding the number of parameters, it is worth noting that the proposed 3D-printed processor has a significantly smaller number of parameters than both FNNs and RNNs. FNNs typically have fewer parameters compared to RNNs, but their performance suffers when modelling temporal dynamics. In contrast, RNNs have a much larger number of parameters. They can capture the temporal dynamics in the data but may suffer from overfitting if the data size is insufficient. The proposed processor offers a middle ground, with a smaller number of parameters than RNNs, while still being able to model temporal dynamics. The competitive performance of the proposed processor with a smaller number of parameters suggests that it may be an efficient and effective solution for tasks requiring the modelling of nonlinear and dynamic systems, especially when the data size is limited. Therefore, the proposed physical processors could be a promising alternative to traditional FNNs and RNNs in specific applications.

Table 5-3. Performance comparison among the proposed 3D-printed processor and software neural networks

Model Name	**Software FNN**	**Software RNN**	**3D-Printed Processor**
Pre-processing	Normalization	Normalization	Gain/scaling
Architectural details	One input layer One hidden layer One output layer	One input layer One hidden layer One output layer	One input layer One hidden layers One output layer
Number of neurons	4+24+1= 29	4+24+1 = 29	4+3*6+6 = 28
Nonlinear function	Sigmoid	Sigmoid	Temperature-dependent resistivity of the material
Accuracy RMSE (%) and R^2 (%)	7.9% and 79.8%	3.4% and 96.7%	4.32% and 95.2%
Number of trainable parameters	145	317	7
Memory capacity	~ 3	~ 6	~ 6
Hardware	CPU	CPU	Hardware

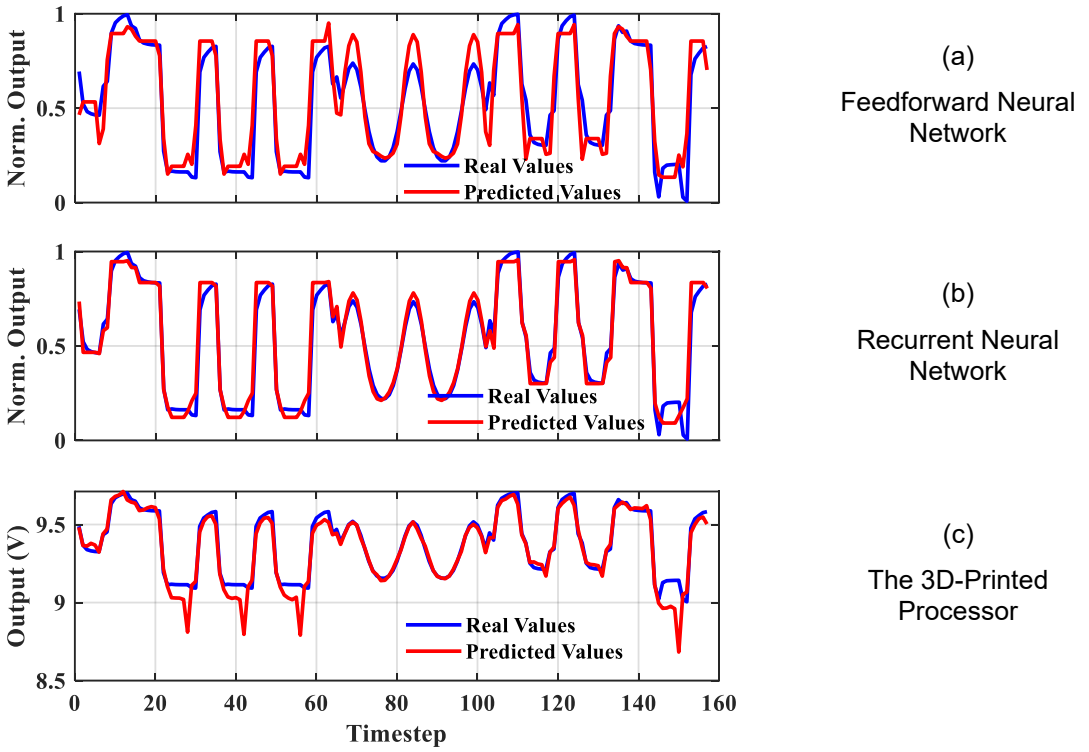


Figure 5.15. Performance comparison of the (a) FNN, (b) RNN, and (c) the proposed 3D printed processor in modelling the NARMA3.

5.6. Summary

In conclusion, the proposed 3D-printed computing platforms using reservoir computing (RC) principles offer a new and innovative approach to sensory data processing. This approach uses conductive filaments, such as carbon-filled PLA, which exhibit interesting time-dependent, nonlinear responses due to self-heating. These responses make 3D-printed resistors suitable for developing RC computers based on ESN topology. By using 3D-printed resistors as nonlinear, coupled physical elements, we have demonstrated the feasibility of creating context from incoming time-series data.

One of the significant advantages of the proposed 3D-printed computing platforms is that they can be designed and fabricated in any facility with access to regular 3D printers. This eliminates the need for sophisticated measurement systems or highly specialized manufacturing processes, making the technology accessible to a broader range of researchers and practitioners. Additionally, the computational capabilities of the printed computer can be increased by modulating control signals, increasing the number of printed neurons, and adjusting the lateral and vertical distances between the neurons.

Another key benefit of the proposed 3D-printed computing platforms is the potential for fully 3D-printed intelligent systems that combine 3D-printed sensors with a 3D-printed contextual computer. This would allow for the instant extraction of context from environmental changes, enabling new possibilities for developing intelligent systems. Furthermore, the principles of this approach can be applied to other manufacturing techniques, such as using coupled temperature-sensitive resistors on silicon chips to process data from micromachined sensors, creating context without digital computations.

The proposed 3D-printed computing platforms have the potential to revolutionize the field of data processing by providing a cost-effective and accessible solution for the development of intelligent systems. This approach can open up new opportunities for researchers and practitioners in the field and pave the way for innovations in the future.

Chapter 6.

Conclusions and Future Work

The proposed 3D-printed computing platform can be useful in many near-sensor applications. These processors can process sensor data in real-time, allowing for real-time monitoring and analysis of various metrics such as movement and temperature. This attribute can enable more advanced applications such as predictive maintenance, quality control, and safety monitoring. One can exploit these processors to drive advanced user interfaces, enabling more immersive and interactive user experiences. If printed and developed in microscales, they will consume less power than traditional processors. Consequently, this allows sensing devices to be powered by small batteries, making them more portable.

The advantages and disadvantages of the proposed computing platform are compared to the existing technologies and summarized in Table 6-1. A CPU, GPU, or FPGA cost can vary widely depending on several factors. CPUs tend to be the most expensive type of hardware, followed by GPUs and FPGAs. However, the cost is not the only consideration when selecting a platform, as other factors such as performance, capabilities, compatibility, and ease of use can also be important. Neuromorphic chips mimic the electrical properties and dynamics of biological neurons and synapses, allowing for low power consumption and high energy efficiency. They can perform multiple computations simultaneously, greatly accelerating the processing of large data sets. Neuromorphic chips can also tolerate high noise levels and errors, making them suitable for harsh environments. However, neuromorphic chips are relatively expensive to produce, which can limit their adoption in some applications. They offer limited scalability due to a fixed number of neurons and synapses and limit the system's scalability. It also suffers from limited programmability, which can limit the system's flexibility.

Physical computing platforms, on the other hand, offer high flexibility. Physical systems developed for reservoir computing can be chosen from a wide range of systems that exhibit nonlinear dynamic behaviour, allowing for increased flexibility regarding the system's properties. These physical systems can be scaled up to larger systems, allowing to processing of larger data sets. These platforms are typically less expensive than

neuromorphic chips. However, the dynamics of these physical systems can be hard to control and may not be as robust to noise and errors as neuromorphic chips. Additionally, these physical systems may be less energy efficient than neuromorphic chips.

The proposed computing platform utilizing 3D-printed processors presents a promising solution for various processing applications. It offers several advantages over traditional manufacturing methods. These include extremely low cost, compatibility with wearable sensors, and the ability to encompass a passive nonlinear dynamical system. Additionally, 3D-printed processors are easily reproducible, allowing for quick and efficient scaling of the platform. Furthermore, the platform is capable of real-time data processing and has the potential for low power consumption and energy efficiency if printed in microscales.

Table 6-1. Advantages and disadvantages of the proposed computing platform compared to the existing solutions

Technology	Advantages	Disadvantages
CPUs	Widely available and easy to program, High performance for general-purpose computing tasks	High power consumption, Less efficient for parallelizable tasks
GPUs	High parallel processing power, Lower power consumption than CPUs for specific tasks	More expensive than CPUs, More challenging to program,
Microcontrollers	Low power consumption, Low cost	Limited computing power and memory, Limited scalability
ASICs	High performance and energy efficiency, Low power consumption	High development cost, Limited flexibility
FPGAs	High performance and energy efficiency, Low power consumption, High flexibility	High development cost, Complex programming
Neuromorphic Computing	High energy efficiency, Low power consumption	High Cost, Limited scalability, Limited programming flexibility
Existing Physical RC Platforms	High performance and energy efficiency, Low power consumption, High flexibility	Limited scalability
Proposed Computing Platform	Extremely low cost, Compatible with 3D-printed sensors, It encompasses a passive nonlinear dynamical system, Easily reproducible, Energy efficient (if developed in microscales)	Possibility of deformation if large signals are applied, Limited control over the dynamics, Limited flexibility and scalability

On the other hand, there are a few limitations to consider with the proposed computing platform. One limitation is the current technology of 3D printing, as the resolution and precision of 3D printed components may not be as high as those produced through traditional manufacturing methods. Additionally, the platform may not be suitable for certain high-performance applications that require very high clock speeds or large amounts of memory. Lastly, the printed processors may deform due to high electrical signals applied to them, which could lead to reduced performance and reliability. Limited expressivity is another challenge that can be addressed to some extent by appropriately selecting the rate at which the input signals are mapped to the physical reservoir within the processors.

6.1. Future Directions

Physical RC is a relatively new field of research, and the future of this field is difficult to predict. However, physical RC will likely continue to be an active area of research as it has the potential to enable the development of more efficient, low-power computing systems for various applications. Physical RC platforms have been applied across multiple research projects and have demonstrated promising results in various tasks, such as time-series prediction, chaotic time-series prediction, and image processing [6], [106]–[108]. However, it is essential to note that these platforms are still in the research and development phase and may not yet be ready for practical applications.

The applicability and feasibility of physical RC platforms depend on various factors, such as the type of platform, the specific application, and the current state of technology. For example, electronic RC platforms are generally easier to implement and control than other physical platforms, such as optical and mechanical RC. However, their performance may be limited by the current state of technology and the devices' quality. Additionally, the devices' cost and the fabrication process's complexity also play a role in the feasibility of these platforms. For example, the fabrication of micro- and nano-electromechanical systems (MEMS and NEMS) is relatively complex and costly, which may limit their scalability and commercial feasibility. Thus, more development and research are needed before these platforms can be easily realized and applied in practical applications.

One potential direction for future research in this field is the development of compatible physical processors with different sensor technologies for power-efficient real-

world applications. This will lead to designing and developing physical reservoir computing platforms for near- or in-sensor computing applications. Researchers can push the boundaries of these computing platforms by addressing the challenges and *knowledge gaps* that still exist in near-sensor computing platforms. By addressing these challenges and knowledge gaps, we can create more effective and efficient near-sensor computing platforms for various applications. Through my Ph.D. research, I have contributed to this field by developing a prototype of a near-sensor computing platform. However, there is much work to make these systems more effective and efficient. Thus, future research may focus on finding ways to make various sensors in each sensory node, such as temperature, humidity, and gas sensors, not only sense the environment but also collaborate to process the data in real time and produce context-aware information. This research can potentially revolutionize neuromorphic computing by significantly contributing to knowledge generation and utilization. This research can also be applied to other areas where *energy efficiency* and *real-time processing* are critical such as robotics, and control systems. Additionally, there is a need to focus on the *design of scalable physical reservoir computing platforms* for large-scale applications.

Specific Research Gap: In many physical reservoirs that have been developed thus far, pre-processing is often required. This pre-processing is often necessary due to adjustments needed for solving specific problems using physical RC, which may differ from software-based approaches. One common requirement is to slow down the introduction of events to the physical RC system, considering the slower timescales associated with physical reservoirs and/or the external implications of the measurement setups. For instance, in the context of photonics-based RC, pre-processing steps may be employed to address the limitations or characteristics of the optical system [109]. Optical systems often have inherent time delays due to various factors such as signal propagation through optical fibers or the response time of optical elements. To account for these delays, pre-processing techniques like introducing artificial time delays or adjusting the timing of events may be necessary. This ensures that the events align appropriately with the dynamics of the physical reservoir. Additionally, optical signals may undergo various forms of distortion or noise during propagation. Pre-processing steps can involve signal conditioning techniques such as filtering, amplification, or equalization to enhance the quality and fidelity of the input signals before they are fed into the physical reservoir. As mentioned earlier, photonics RC often utilizes intensity or phase modulation schemes.

Pre-processing may involve choosing the appropriate modulation technique, optimizing the modulation parameters, or designing specific modulation waveforms to ensure efficient signal encoding and decoding within the physical reservoir. Last but not least, physical reservoirs may require calibration and synchronization procedures to align the components, optimize the system parameters, and mitigate any non-idealities. This may involve adjusting the optical power levels, aligning optical elements, or compensating for nonlinearities introduced by the physical system. Therefore, the specific requirements and techniques may vary depending on the characteristics of the physical system, the targeted application, and the desired performance of the physical reservoir. Pre-processing at the current level of research and development on physical RCs, plays a crucial role in adapting the input signals and the measurement setup to the specific characteristics and constraints of physical reservoirs, enabling them to effectively solve a wide range of computational tasks. The next research question can be: how can we eliminate the demand for pre-processing in physical RCs?

Specific Applications: One potential direction for future research in this field is the integration of the physical reservoir computing platforms in Internet of Things (IoT) devices to promote sustainability in smart cities. Researchers can focus on developing and using unconventional processors to perform in-memory computing and identify potential applications in various smart city contexts, such as monitoring the environment and public safety. Another area of further research could be developing intelligent systems to address the existing challenges. One example is the detection of specific gas concentrations in a gas mixture, which is of great importance in industrial, residential, or agricultural applications. Existing systems, often called electronic noses, utilize several gas sensors and sophisticated algorithms to overcome cross-sensitivity issues, making such systems prohibitively expensive, power-hungry, and large for wide-scale applications. Researchers can develop a cognizant gas sensor to address these concerns and apply their findings to other sensing applications with similar challenges, such as e-tongue or biomedical sensing. Researchers can also focus on addressing a challenge existing gas sensing systems face: the need for continuous communication of the gas content for processing purposes. While it is crucial to quickly and accurately detect gas concentrations, constantly transmitting this information is not always necessary. The system can reduce power consumption and improve efficiency by processing data locally and only communicating the necessary information.

On-Site Decision Making: Continuous data communication for processing purposes may not be necessary for all applications. The system can reduce power consumption and improve efficiency by processing data locally and only communicating the necessary information. This goal can be achieved using physical reservoir computing systems that perform real-time data processing and on-site decision-making without constant communication with a central processing unit. All the potential future directions are summarized in a mind map shown in Figure 6.1.

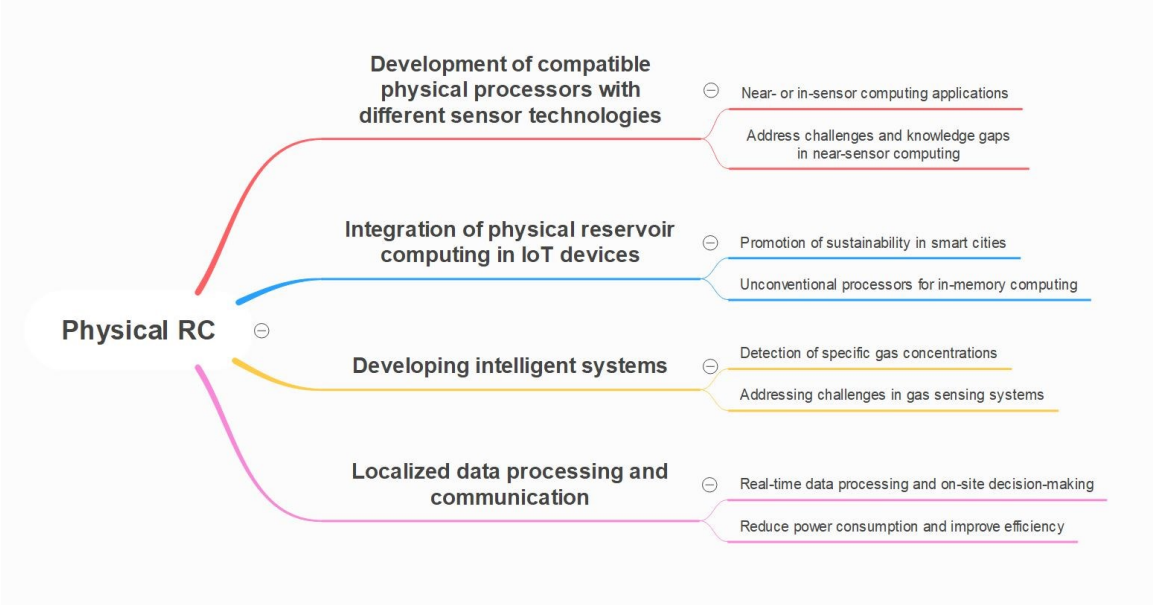


Figure 6.1. Future directions.

In conclusion, physical RC is a relatively new field of research with great potential to revolutionize processing systems by enabling the development of more efficient and low-power computing systems for various applications. Despite the promising results in multiple tasks, physical RC platforms are still in the research and development phase and may not yet be ready for practical applications. However, continued research and development in this field will pave the way for creating more efficient and powerful processing systems for various applications. Additionally, fostering collaborations between researchers from different areas, such as physics, computer science, electrical engineering, and materials science, will accelerate the development and application of physical RC systems.

References

[1] Y. Yang, “Multi-tier computing networks for intelligent IoT,” *Nat. Electron.*, vol. 2, no. 1, pp. 4–5, 2019.

[2] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, “Energy conservation in wireless sensor networks: A survey,” *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009.

[3] F. Zhou and Y. Chai, “Near-sensor and in-sensor computing,” *Nat. Electron.*, vol. 3, no. 11, pp. 664–671, 2020, doi: 10.1038/s41928-020-00501-9.

[4] Z. Konkoli, “On developing theory of reservoir computing for sensing applications: the state weaving environment echo tracker (SWEET) algorithm,” *Int. J. Parallel, Emergent Distrib. Syst.*, vol. 33, no. 2, pp. 121–143, 2018, doi: 10.1080/17445760.2016.1241880.

[5] M. Lukoševičius, H. Jaeger, and B. Schrauwen, “Reservoir computing trends,” *KI - Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[6] Z. Tong, “Reservoir computing with untrained convolutional neural networks for image recognition,” in *24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 1289–1294.

[7] G. Holzmann, “Reservoir computing: A powerful black-box framework for nonlinear audio processing,” in *Proceedings of the 12th International Conference on Digital Audio Effects, DAFx 2009*, 2009, pp. 90–97.

[8] M. Santhosh, “Current advances and approaches in wind speed and wind power forecasting for improved renewable energy integration : A review,” vol. 2, no. 6, pp. e121781-20, 2020, doi: 10.1002/eng2.12178.

[9] Q. M. Saleh, “Design of a neuromemristive echo state network architecture,” Rochester Institute of Technology, 2015.

[10] P. Antonik, A. Smerieri, F. Duport, M. Haelterman, and S. Massar, “FPGA implementation of reservoir computing with online learning,” in *Belgian-Dutch Conference on Machine Learning*, 2015, [Online]. Available: http://homepage.tudelft.nl/19j49/benelearn/papers/Paper_Antonik.pdf.

[11] H. O. Sillin *et al.*, “A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing,” *Nanotechnology*, vol. 24, no. 38, p. 384004, 2013.

[12] B. Barazani, G. Dion, J. F. Morissette, L. Beaudoin, and J. Sylvestre, “Microfabricated neuroaccelerometer: integrating sensing and reservoir computing in MEMS,” *J. Microelectromechanical Syst.*, vol. 29, no. 3, pp. 338–347, 2020.

[13] T. T. Rogers and J. L. McClelland, “Parallel distributed processing at 25: Further explorations in the microstructure of cognition,” *Cogn. Sci.*, vol. 38, no. 6, pp. 1024–1077, Aug. 2014, doi: https://doi.org/10.1111/cogs.12148.

[14] M. Bonfim, R. Roque, E. Coutinho, K. Dias, and S. Fernandes, “Identifying performance bottlenecks in software data planes for cloud-based NFV services,” in *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, 2018, pp. 1–7, doi: 10.1109/NOMS.2018.8406161.

[15] J. Von Neumann, *First draft of a report on the EDVAC*. Moore School of Electrical Engineering, University of Pennsylvania, 1945.

[16] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, “Opportunities for neuromorphic computing algorithms and applications,” *Nat. Comput. Sci.*, vol. 2, no. 1, pp. 10–19, 2022, doi: 10.1038/s43588-021-00184-y.

[17] M. Davies *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[18] P. A. Merolla *et al.*, “Network and interface,” *Sciencemag.Org*, vol. 345, no. 7812, pp. 668–673, 2014.

[19] H. P. Graf and L. D. Jackel, “Analog electronic neural network circuits,” *IEEE Circuits Devices Mag.*, vol. 5, no. 4, pp. 44–49, 1989, doi: 10.1109/101.29902.

[20] H. C. Card, “Analog VLSI Neural Learning Circuits — A Tutorial BT - VLSI for Neural Networks and Artificial Intelligence,” J. G. Delgado-Frias and W. R. Moore, Eds. Boston, MA: Springer US, 1994, pp. 1–23.

[21] A. Rodan and P. Tiňo, “Minimum complexity echo state network,” *IEEE Trans. Neural Networks*, vol. 22, no. 1, pp. 131–144, 2011.

[22] M. Verleysen, “Advances in computational intelligence and learning,” in *15th European Symposium on Artificial Neural Networks ; ESANN 2007*, 2007, no. April, pp. 25–27.

[23] G. Tanaka *et al.*, “Recent advances in physical reservoir computing: A review,” *Neural Networks*, vol. 115, pp. 100–123, 2019.

[24] J. C. Coulombe, M. C. A. York, and J. Sylvestre, “Computing with networks of nonlinear mechanical oscillators,” *PLoS One*, vol. 12, no. 6, p. e0178663, 2017.

[25] G. Dion, S. Mejaouri, and J. Sylvestre, “Reservoir computing with a single delay-coupled non-linear mechanical oscillator,” *J. Appl. Phys.*, vol. 124, p. 152132, 2018.

[26] F. Caravelli and J. Carbajal, “Memristors for the curious outsiders,” *Technologies*, vol. 6, no. 4, p. 118, 2018.

[27] M. J. Marinella and S. Agarwal, “Efficient reservoir computing with memristors,” *Nat. Electron.*, vol. 2, no. 10, pp. 437–438, 2019.

[28] J. Moon *et al.*, “Temporal data classification and forecasting using a memristor-based reservoir computing system,” *Nat. Electron.*, vol. 2, no. 10, pp. 480–487, 2019.

[29] R. Midya *et al.*, “Reservoir computing using diffusive memristors,” *Adv. Intell. Syst.*, vol. 1, no. 7, p. 1900084, 2019.

[30] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu, “Reservoir computing using dynamic memristors for temporal information processing,” *Nat. Commun.*, vol. 8, p. 2204, 2017.

[31] V. Athanasiou and Z. Konkoli, “On using reservoir computing for sensing applications: exploring environment-sensitive memristor networks,” *Int. J. Parallel, Emergent Distrib. Syst.*, vol. 33, no. 4, pp. 367–386, 2018.

[32] M. S. Kulkarni, “Memristor-based reservoir computing,” Portland State University, 2012.

[33] K. Vandoorne *et al.*, “Experimental demonstration of reservoir computing on a silicon photonics chip,” *Nat. Commun.*, vol. 5, p. 3541, 2014.

[34] F. Duport, A. Smerieri, A. Akrout, M. Haelterman, and S. Massar, “Fully analogue photonic reservoir computer,” *Sci. Rep.*, vol. 6, p. 22381, 2016.

[35] A. Lugnan *et al.*, “Photonic neuromorphic information processing and reservoir computing,” *APL Photonics*, vol. 5, p. 020901, 2020.

[36] L. Larger *et al.*, “Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing,” *Opt. Express*, vol. 20, no. 3, p. 3241, 2012.

[37] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer, “Parallel photonic information processing at gigabyte per second data rates using transient states,” *Nat. Commun.*, vol. 4, p. 1364, 2013.

[38] A. N. Tait, M. A. Nahmias, B. J. Shastri, and P. R. Prucnal, “Broadcast and weight: An integrated network for scalable photonic spike processing,” *J. Light. Technol.*, vol. 32, no. 21, pp. 4029–4041, 2014.

[39] Y. Paquot, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, “Reservoir computing: a photonic neural network for information processing,” *Nonlinear Opt. Appl. IV*, vol. 7728, p. 77280B, 2010.

[40] Q. Vinckier *et al.*, “High-performance photonic reservoir computer based on a coherently driven passive cavity,” *Optica*, vol. 2, no. 5, p. 438, 2015.

[41] O. R. Lykkebø, S. Harding, G. Tufte, and J. F. Miller, “Mecobo: A hardware and software platform for in materio evolution,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8553 LNCS, pp. 267–279, 2014.

[42] M. Dale, S. Stepney, J. F. Miller, and M. Trefzer, “Reservoir computing in materio: A computational framework for in materio computing,” in *Proceedings of the International Joint Conference on Neural Networks*, 2017, pp. 2178–2185.

[43] M. N. DALE, “Reservoir computing in materio,” University of York, 2018.

[44] K. Nakajima, H. Hauser, T. Li, and R. Pfeifer, “Information processing via physical soft body,” *Sci. Rep.*, vol. 5, p. 10487, 2015.

[45] V. Privman, “Biomolecular computing: Learning through play,” *Nat. Nanotechnol.*, vol. 5, no. 11, pp. 767–768, 2010.

[46] A. Adamatzky, *Advances in physarum machines*, 1st ed. Springer International Publishing, 2016.

[47] Andrew Adamatzky, B. D. L. Asai, and Costello Tetsuya, *Reaction-diffusion computers*, 1st ed. Elsevier Science, 2005.

[48] M. K. Massey *et al.*, “Computing with carbon nanotubes: Optimization of threshold logic gates using disordered nanotube/polymer composites,” *J. Appl. Phys.*, vol. 117, no. 13, p. 134903, 2015.

[49] K. Nakajima, H. Hauser, T. Li, and R. Pfeifer, “Exploiting the dynamics of soft materials for machine learning,” *Soft Robot.*, vol. 5, no. 3, pp. 339–347, 2018.

[50] V. Shirmohammadli and B. Bahreyni, “Development of a thermo-computing platform,” in *2021 21st International Conference on Solid-State Sensors, Actuators and Microsystems (Transducers)*, 2021, pp. 1307–1310.

[51] M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann, “Hands-on reservoir computing: a tutorial for practical implementation,” *Neuromorphic Comput. Eng.*, vol. 2, no. 3, 2022, doi: 10.1088/2634-4386/ac7db7.

[52] P. Bhovad and S. Li, “Physical reservoir computing with origami and its application to robotic crawling,” *Sci. Rep.*, vol. 11, p. 13002, 2021, doi: 10.1038/s41598-021-92257-1.

[53] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[54] M. L. Alomar Barceló, “Methodologies for hardware implementation of reservoir computing systems,” Universitat de les Illes Balears, 2017.

[55] H. Jaeger, “The ‘ echo state ’ approach to analysing and training recurrent neural networks,” *Tech. Rep. GMD Rep. 148, Ger. Natl. Res. Cent. forInformation Technol.*, no. 148, pp. 1–47, 2001.

[56] H. Jaeger, “The ‘echo state’ approach to analysing and training recurrent neural networks-with an erratum note,” *Bonn, Ger. Ger. Natl. Res. Cent. Inf. Technol. GMD Tech. Rep.*, vol. 148, Jan. 2001.

[57] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, 2002.

[58] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[59] A. Stockdill, “Neuromorphic computing with reservoir neural networks on memristive hardware,” University of Canterbury, 2016.

[60] H. Jaeger, “Short term memory in echo state networks,” *GMD Rep. 152*, no. 152, pp. 1–60, 2002.

[61] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, “Optimization and applications of echo state networks with leaky- integrator neurons,” *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

[62] L. Appeltant *et al.*, “Information processing using a single dynamical node as complex system,” *Nat. Commun.*, vol. 2, p. 468, 2011.

[63] D. Verstraeten, B. Schrauwen, and D. Stroobandt, “Reservoir computing with stochastic bitstream neurons,” in *Proceedings of the 16th Annual ProRISC Workshop*, 2005, pp. 454–459.

[64] Y. Paquot *et al.*, “Optoelectronic reservoir computing,” *Sci. Rep.*, vol. 2, p. 287, 2012.

[65] H. Soh and Y. Demiris, “Iterative temporal learning and prediction with the sparse online echo state gaussian process,” in *Proceedings of the International Joint Conference on Neural Networks*, 2012, pp. 10–15.

[66] A. Jalalvand, G. Van Wallendael, and R. Van De Walle, “Real-time reservoir computing network-based systems for detection tasks on visual contents,” in *Proceedings - 7th International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2015*, 2015, pp. 146–151.

[67] H. Jaeger, “Adaptive nonlinear system identification with Echo State networks,” in *Proceedings of the 15th International Conference on Neural Information Processing Systems*, 2002, pp. 609–616.

[68] H. Jaeger, “Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach,” *GMD-Forschungszentrum Informationstechnik, 2002.*, vol. 5, 2002.

[69] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass, “The role of feedback in morphological computation with compliant bodies,” *Biol. Cybern.*, vol. 106, no. 10, pp. 595–613, 2012.

[70] H. Jaeger and H. Haas, “Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication,” *Science (80-.).*, vol. 304, no. 5667, pp. 78–80, 2004.

[71] N. Bertschinger and T. Natschläger, “Real-time computation at the edge of chaos in recurrent neural networks,” *Neural Comput.*, vol. 16, no. 7, pp. 1413–1436, 2004.

[72] F. Stelzer, A. Röhm, K. Lüdge, and S. Yanchuk, “Performance boost of time-delay reservoir computing by non-resonant clock cycle,” *Neural Networks*, vol. 124, pp. 158–169, 2020.

[73] M. R. Salehi and L. Dehyadegari, “Toward optical signal processing using photonic reservoir computing,” *J. Mod. Opt.*, vol. 16, no. 15, p. 111, 2008.

[74] K. Vandoorne, J. Dambre, D. Verstraeten, B. Schrauwen, and P. Bienstman, “Parallel reservoir computing using optical amplifiers,” *IEEE Trans. Neural Networks*, vol. 22, no. 9, pp. 1469–1481, 2011.

[75] F. Duport, B. Schneider, A. Smerieri, M. Haelterman, and S. Massar, “All-optical reservoir computing,” *Opt. Express*, vol. 20, no. 20, pp. 1958–1964, 2012.

[76] A. Dejonckheere *et al.*, “All-optical reservoir computer based on saturation of absorption,” *Opt. Express*, vol. 22, no. 9, pp. 10868–10881, 2014.

[77] C. Verma and J. Singh, “Memristor-the missing circuit element,” *Nanoelectron. Devices Hardw. Softw. Secur.*, vol. CT-18, no. 5, pp. 507–519, 1971.

[78] M. N. Ashner, U. Paudel, M. Luengo-Kovac, J. Pilawa, T. J. Shaw, and G. C. Valley, “Photonic reservoir computer with all-optical reservoir,” in *SPIE 11703, 117030L*, 2021, pp. 1–12.

[79] D. Brunner and I. Fischer, “Reconfigurable semiconductor laser networks based on diffractive coupling,” *Opt. Lett.*, vol. 40, no. 16, p. 3854, 2015.

[80] A. Sattar, S. Fostner, and S. A. Brown, “Quantized conductance and switching in percolating nanoparticle films,” *Phys. Rev. Lett.*, vol. 111, no. 13, p. 136808, 2013.

[81] A. Z. Stieg, A. V. Avizienis, H. O. Sillin, C. Martin-Olmos, M. Aono, and J. K. Gimzewski, “Emergent criticality in complex Turing B-type atomic switch networks,” *Adv. Mater.*, vol. 24, pp. 286–293, 2012.

[82] K. Nakajima and I. Fischer, *Reservoir computing theory, physical implementations, and applications*. Springer, 2021.

[83] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, “A substrate-independent framework to characterize reservoir computers,” in *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2019, vol. 475, p. 20180723, doi: 10.1098/rspa.2018.0723.

[84] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, “A substrate-independent framework to characterize reservoir computers,” in *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2019, vol. 475, no. 20180723, doi: 10.1098/rspa.2018.0723.

[85] S. Hafizovic *et al.*, “A CMOS-based microelectrode array for interaction with neuronal cultures,” *J. Neurosci. Methods*, vol. 164, no. 1, pp. 93–106, 2007.

[86] K. P. Dockendorf, I. Park, P. He, J. C. Príncipe, and T. B. DeMarse, “Liquid state machines and cultured cortical networks: The separation property,” *BioSystems*, vol. 95, no. 2, pp. 90–97, 2009.

[87] M. R. Dranias, H. Ju, E. Rajaram, and A. M. J. VanDongen, “Short-term memory in networks of dissociated cortical neurons,” *J. Neurosci.*, vol. 33, no. 5, pp. 1940–1953, 2013.

[88] B. Jones, D. Stekel, J. Rowe, and C. Fernando, “Is there a liquid state machine in the bacterium Escherichia coli?,” in *Proceedings of the 2007 IEEE Symposium on Artificial Life, CI-ALife 2007*, 2007, pp. 187–191.

[89] F. Jazaeri, A. Beckers, A. Tajalli, and J.-M. Sallese, “A Review on Quantum Computing: Qubits, Cryogenic Electronics and Cryogenic MOSFET Physics,” *arXiv Quantum Phys.*, Aug. 2019, [Online]. Available: http://arxiv.org/abs/1908.02656.

[90] O. Obst *et al.*, “Nano-scale reservoir computing,” *Nano Commun. Netw.*, vol. 4, no. 4, pp. 189–196, 2013.

[91] V. Shirmohammadli and B. Bahreyni, “A neuromorphic electrothermal processor for near-sensor computing,” *Adv. Mater. Technol.*, vol. 7, no. 11, p. 2200361, 2022, doi: 10.1002/admt.202200361.

[92] A. Ü. Keskin, “A simple analog behavioural model for NTC thermistors including selfheating effect,” *Sensors Actuators A Phys.*, vol. 118, no. 2, pp. 244–247, 2005, doi: https://doi.org/10.1016/j.sna.2004.06.034.

[93] A. J. Capel, R. P. Rimington, M. P. Lewis, and S. D. R. Christie, “3D printing for chemical, pharmaceutical and biological applications,” *Nat. Rev. Chem.*, vol. 2, no. 12, pp. 422–436, 2018, doi: 10.1038/s41570-018-0058-y.

[94] M. R. Khosravani and T. Reinicke, “3D-printed sensors: Current progress and future

challenges," *Sensors Actuators A Phys.*, vol. 305, p. 111916, 2020, doi: https://doi.org/10.1016/j.sna.2020.111916.

[95] T. N. Mangoma, S. Yamamoto, G. G. Malliaras, and R. Daly, "Hybrid 3D/inkjet-printed organic neuromorphic transistors," *Adv. Mater. Technol.*, vol. 7, p. 2000798, 2022, doi: 10.1002/admt.202000798.

[96] Y. Tuchman *et al.*, "Organic neuromorphic devices: Past, present, and future challenges," *MRS Bulletin*, vol. 45, no. 8. pp. 619–630, 2020, doi: 10.1557/mrs.2020.196.

[97] C. Bao, T. H. Kim, A. Hassanpoor Kalhori, and W. S. Kim, "A 3D-printed neuromorphic humanoid hand for grasping unknown objects," *iScience*, vol. 25, no. 4, p. 104119, 2022, doi: 10.1016/j.isci.2022.104119.

[98] C. Wan *et al.*, "An artificial sensory neuron with tactile perceptual learning," *Adv. Mater.*, vol. 30, no. 30, p. 1801291, 2018, doi: 10.1002/adma.201801291.

[99] X. Lin *et al.*, "All-optical machine learning using diffractive deep neural networks," *Science (80-.).*, vol. 361, pp. 1004–1008, 2018.

[100] P. F. Flowers, C. Reyes, S. Ye, M. J. Kim, and B. J. Wiley, "3D printing electronic components and circuits with conductive thermoplastic filament," *Addit. Manuf.*, vol. 18, pp. 156–163, 2017, doi: https://doi.org/10.1016/j.addma.2017.10.002.

[101] Y. A. Cengel, *Heat transfer: a practical approach*, 2nd ed. New York: McGraw-Hill, 2004.

[102] V. Shirmohammadli and B. Bahreyni, "A 3D-printed computer," *Adv. Intell. Syst.*, vol. 5, p. 2300015, 2023, doi: 10.1002/aisy.202300015.

[103] M. Inubushi and K. Yoshimura, "Reservoir computing beyond memory-nonlinearity trade-off," *Sci. Rep.*, vol. 7, p. 10199, 2017, doi: 10.1038/s41598-017-10257-6.

[104] "STEVAL-STLKT01V1 - SensorTile development kit - STMicroelectronics." https://www.st.com/en/evaluation-tools/steval-stlkt01v1.html.

[105] "Green500." https://www.top500.org/lists/green500/2023/06/.

[106] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Comput.*, vol. 22, no. 12, pp. 3207–3220, 2010, doi: 10.1162/NECO_a_00052.

[107] Francis Wyffels, B. Schrauwen;, and D. Stroobandt, "System modeling with reservoir computing," in *Wyffels2008SystemMW*, 2008, [Online]. Available: https://api.semanticscholar.org/CorpusID:114159678.

[108] J. Wang, T. Niu, G. S. Member, H. Lu, and S. Member, "A novel framework of

reservoir computing for deterministic and probabilistic wind power forecasting," vol. 11, no. 1, pp. 337–349, 2020.

[109] G. Van Der Sande, D. Brunner, and M. C. Soriano, "Advances in photonic reservoir computing," *Nanophotonics*, vol. 6, no. 3, pp. 561–576, 2017.

[110] "Ultimaker 3." https://ultimaker.com/3d-printers/ultimaker-3 (accessed Mar. 22, 2023).

Appendix A.

Materials and Methods

3D Printing Setup

Fused deposition modelling (FDM) is the most widely used 3D printing technology. It uses a filament spool fed to an extrusion head with a heated nozzle. Once the extrusion head heats, it softens and lays down the heated material at set locations, where it cools to create a material layer. The nozzle then moves down to deposit the next layer. The reservoir is printed with a double extrusion 3D printer (Ultimaker 3) as shown in Figure A.1 [110]. The material is printed with a default nozzle size of 0.4 mm. The printer settings were adjusted to have a layer height of 0.1 mm for each printed layer and a wall thickness of 0.8 mm. Thus, the minimum feature size for the resistor is 0.8 mm in the *x* and *y* directions, while it is 0.1 mm in the *z*-direction.

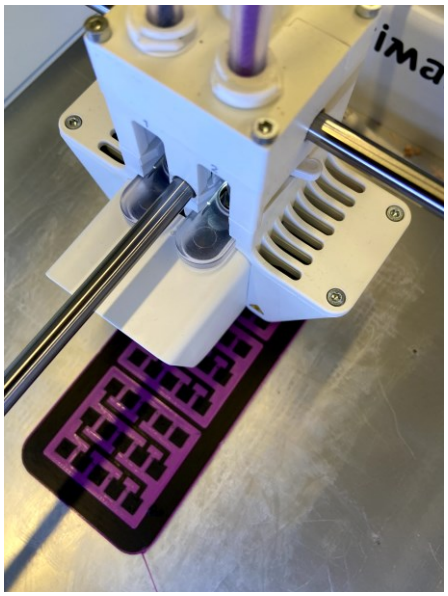


Figure A.1. 3D printing setup.

The key strength of FDM is the availability of a wide range of materials, including thermoplastics such as Polylactic Acid (PLA), a vegetable-based, biodegradable thermoplastic. PLA is an electrical insulator, so it was used as the substrate and insulating material between the conductive layers. The resistors were printed using carbon-PLA composite filaments. We used 2.85 mm pure PLA filament (Ultrafuse series from BASF)

for the main structure and 2.85 mm carbon-PLA composite (RM-PL0100 from Lulzbot) for the conductive segments. Before 3D printing, the conductive PLA filament has a volume resistivity of 15 Ω-cm. A 3D printed structure in the *x*- or *y*-directors has a resistivity of 30 Ω-cm. 3D printed layers printed in the *z*-direction have a resistivity of 115 Ω-cm. Multi-layer resistors offer higher conductivity as more conduction paths will be possible due to the increased connections between carbon elements within this structure. Silver conductive paste/epoxy was utilized to attach wires to the printed contacts, which were cured on a hot plate at 50 ℃ for 1 hour.

Evaluation of the Analytical Model

COMSOL Multiphysics® 5.5 software was used to perform the heat transfer modelling. COMSOL Multiphysics is a simulation software that can be used to model and analyze a wide range of physical systems, including the self-heating of a resistor. The software combines multiple physics modules and allows users to create custom models and simulations using a user-friendly graphical interface. To model self-heating in a resistor using COMSOL Multiphysics, we select the appropriate physics modules, such as electric currents, heat transfer in solids, and coupled physics like electromagnetic heating, and define the material properties of the resistor and surrounding environment. We also determine the geometry of the resistor and the thermal and electrical boundary conditions, such as the applied current and temperature at the boundaries. Once the model is set up, the software uses numerical methods to solve the governing equations and simulate the system's behaviour over time. Specifically, several physics modules must be combined to simulate and model self-heating in a resistor. Two physics were employed here; electric currents (ec) from the AC/DC module and heat transfer in solids from the Heat Transfer module, and a coupled physics (*i.e.,* Multiphysics) known as electromagnetic heating to study the effect of self-heating in the under-study electrothermally conductive material, conductive PLA.

The boundary condition of the bottom of the device is set to a specific temperature, *i.e.,* room temperature. This is often referred to as a temperature boundary condition and is used to specify the temperature at a particular surface or location in a simulation. All other surfaces are set to be thermally isolated. Materials have been added manually with the values summarized in Table A.1. The device's dimensions are mentioned in Table A.2. We use these dimensions unless otherwise stated.

Table A.1. Material Characteristics

Material	T_g (K)	T_g (K)	ρ_m (kg/m^3)	C_p (J/kg.K)	k (W/m.K)	ϵ_r	ρ_e (Ω-cm)
PLA	60 - 65	180 - 220	1210 -1240	1180 - 1210	0.12 – 0.15	2.88 – 3.48	-
Conductive PLA	60 - 70	170	1220	1180	0.18 – 0.2	-	30 - 115

Table A.2. 3D-Printed Processor Specifications

Parameter	Definition	Value	Minimum (Limit)
W	Width of the resistor	1.6 [mm]	0.8 [mm]
th	Thickness of the resistor	0.5 [mm]	0.1 [mm]
L	Length of the resistor	4.8 [mm]	0.8 [mm]
W_Electrode	Width of the electrodes	5 [mm]	5 [mm]
gap	Spacing between resistor	0.5 [mm]	0.1 [mm]

Deriving the Analytical Model

To solve the heat transfer problem for resistors/neurons in a regular PLA, it is indeed essential to start with solving two sets of problems: one for the resistor/neuron and the second for the regular PLA. Therefore, we will need four types of boundary conditions. For neuron/resistor 1, we will need to solve

$$\frac{1}{r}\frac{d}{dr}\left(\kappa_c\, r\frac{dT_r}{dr}\right) + \dot{e}_{gen} = 0$$
$$\frac{1}{r}\frac{d}{dr}\left(\kappa_{PLA} r\frac{dT_{PLA}}{dr}\right) = 0 \qquad \text{(A-1)}$$

where κ_c is the thermal conductivity of the conductive PLA and $\dot{e}_{\text{gen}}$ is the generated heat in the resistor per volume. The set of boundary conditions is given as:

$$T_{PLA}(r)|_{r=t_{PLA}} = T_a$$
$$T_{PLA}(r)|_{r=r_1} = T_r(r)|_{r=r_1} \qquad \text{(A-2)}$$

$$-\kappa_{PLA}\left.\frac{dT_{PLA}}{dr}\right|_{r=r_1} = -\kappa_c\left.\frac{dT_r}{dr}\right|_{r=r_1} + \dot{e}_{gen,1}$$

$$\kappa_c\left.\frac{dT_r}{dr}\right|_{r=0} = 0$$

The first is the specific temperature boundary condition, the second and third are the interface boundary conditions at the interface of the resistor/neuron and the PLA, and the last is the thermal symmetry boundary condition. Solving Eq. (A-1) for resistor 1 considering the boundary conditions in Eq. (A-2), gives the temperature distribution around resistor/neuron 1 due to its self-heating as

$$\Delta T_1(r) = \begin{cases} \frac{\rho_e J_1^2 r_1^2}{\kappa_{PLA}}\left(\frac{\kappa_{PLA}}{4\kappa_c}\left(1-\left(\frac{r}{r_1}\right)^2\right)+\mathrm{Ln}\left(\frac{t_{PLA}}{r_1}\right)\right) & r \le r_1 \\ \frac{\rho_e J_1^2 r_1^2}{\kappa_{PLA}}\mathrm{Ln}\left(\frac{t_{PLA}}{r}\right) & r_1 < r < t_{PLA} \\ 0 & r \ge t_{PLA} \end{cases} \quad \text{(A-3)}$$

where $\Delta T_i(r) = T_i(r) - T_a$. The equation above results in $q_1 = -\kappa_{PLA}\left.\frac{dT}{dr}\right|_{r=r_1} = \rho_e {J_1}^2 r_1$. This heat flux is used in the interface of the resistor/neuron and the PLA at $r = r_1$ as a boundary condition in the main context. One important note is the negligible temperature deviations inside the resistor if the resistor's dimensions are comparably smaller than the dimensions of the PLA.

For resistor/neuron 3, due to its self-heating, we have

$$\Delta T_3(r) = \begin{cases} \frac{\rho_e J_3^2 r_3^2}{\kappa_{PLA}}\left(\frac{\kappa_{PLA}}{4\kappa_c}\left(1-\left(\frac{r}{r_3}\right)^2\right)+\mathrm{Ln}\left(\frac{{t_{PLA}}^2-{R_d}^2}{2\,r_3 R_d}\right)\right) & r \le r_3 \\ \frac{\rho_e J_3^2 r_3^2}{\kappa_{PLA}}\mathrm{Ln}\left(\frac{{t_{PLA}}^2-{R_d}^2}{r\,r'}\right) & r_3 < r < t_{PLA}-R_{13} \\ 0 & r \ge t_{PLA}-R_{13} \end{cases} \quad \text{(A-4)}$$

where $q_3 = -\kappa_{PLA}\left.\frac{dT}{dr}\right|_{r=r_3} = \rho_e {J_3}^2 r_3$. To go back to the Cartesian coordinate system and keep its origin at the center of resistor 1, we replace $r = \sqrt{y^2+z^2}$ in Eq. (A-3). Whereas, for resistor 3 in Eq. (A-4), we replace $r = \sqrt{y^2+(z-R_d)^2}$ and $r' = \sqrt{y^2+(z+R_d)^2}$. These changes result in

$$\Delta T_1(y,z)=\begin{cases}\dfrac{\rho_e J_1^2 r_1^2}{\kappa_{PLA}}\left(\dfrac{\kappa_{PLA}}{4\kappa_c}\left(1-\dfrac{y^2+z^2}{r_1^2}\right)+\mathrm{Ln}\left(\dfrac{t_{PLA}}{r_1}\right)\right) & \sqrt{y^2+z^2}\le r_1\\ \dfrac{\rho_e J_1^2 r_1^2}{\kappa_{PLA}}\mathrm{Ln}\left(\dfrac{t_{PLA}}{\sqrt{y^2+z^2}}\right) & r_1<\sqrt{y^2+z^2}<t_{PLA}\\ 0 & \sqrt{y^2+z^2}\ge t_{PLA}\end{cases}$$

$$\Delta T_3(y,z)=\begin{cases}\dfrac{\rho_e J_3^2 r_3^2}{\kappa_{PLA}}\left(\dfrac{\kappa_{PLA}}{4\kappa_c}\left(1-\dfrac{y^2+z^2+{R_d}^2}{r_3^2}\right)+\mathrm{Ln}\left(\dfrac{{t_{PLA}}^2-{R_d}^2}{2\,r_3R_d}\right)\right) & \sqrt{y^2+(z-R_d)^2}\le r_3\\ \dfrac{\rho_e J_3^2 r_3^2}{\kappa_{PLA}}\mathrm{Ln}\left(\dfrac{{t_{PLA}}^2-{R_d}^2}{\sqrt{y^2+(z-R_d)^2}\sqrt{y^2+(z+R_d)^2}}\right) & r_3<\sqrt{y^2+(z-R_d)^2}<t_{PLA}-R_d\\ 0 & \sqrt{y^2+(z-R_d)^2}\ge t_{PLA}-R_d\end{cases}$$

(A-5)

Therefore, for resistors 2 and 4, by replacing $y\ =y\ -R_{12}$ and $y\ =y\ -R_{34}$ we have

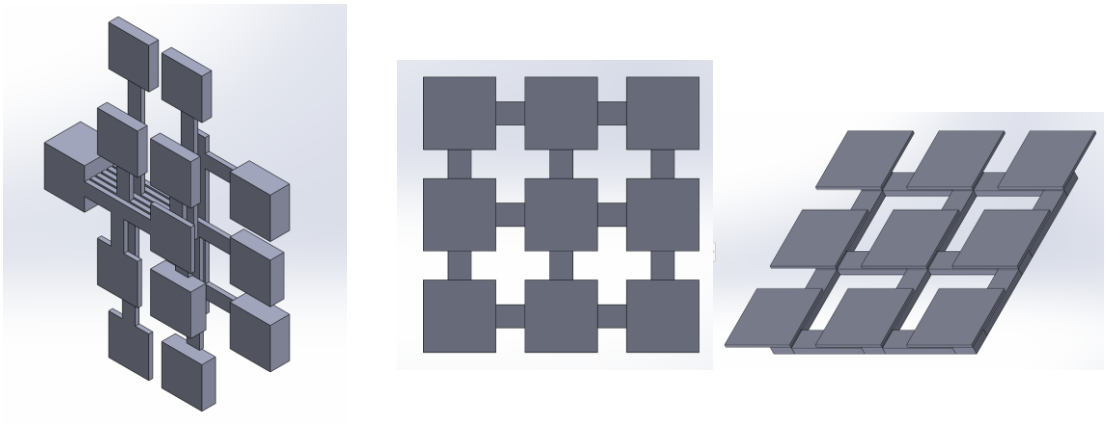
$$\Delta T_2(y,z)=\begin{cases}\dfrac{\rho_e J_2^2 r_2^2}{\kappa_{PLA}}\left(\dfrac{\kappa_{PLA}}{4\kappa_c}\left(1-\dfrac{(y\ -R_{12})^2+z^2}{r_2^2}\right)+\mathrm{Ln}\left(\dfrac{t_{PLA}}{r_2}\right)\right) & \sqrt{(y\ -R_{12})^2+z^2}\le r_2\\ \dfrac{\rho_e J_2^2 r_2^2}{\kappa_{PLA}}\mathrm{Ln}\left(\dfrac{t_{PLA}}{\sqrt{(y\ -R_{12})^2+z^2}}\right) & r_2<\sqrt{(y\ -R_{12})^2+z^2}<t_{PLA}\\ 0 & \sqrt{(y\ -R_{12})^2+z^2}\ge t_{PLA}\end{cases}$$

(A-6)

$$\Delta T_4(y,z) = \begin{cases} \dfrac{\rho_e J_4^2 r_4^2}{\kappa_{PLA}}\left(\dfrac{\kappa_{PLA}}{4\kappa_c}\left(1-\dfrac{(y-R_{34})^2+z^2+{R_d}^2}{r_4^2}\right)+\mathrm{Ln}\left(\dfrac{{t_{PLA}}^2-{R_d}^2}{2\,r_4 R_d}\right)\right) & \sqrt{(y-R_{34})^2+(z-R_d)^2}\le r_4 \\ \dfrac{\rho_e J_4^2 r_4^2}{\kappa_{PLA}}\mathrm{Ln}\left(\dfrac{{t_{PLA}}^2-{R_d}^2}{\sqrt{(y-R_{34})^2+(z-R_d)^2}\sqrt{(y-R_{34})^2+(z+R_d)^2}}\right) & r_4<\sqrt{(y-R_{34})^2+(z-R_d)^2}<t_{PLA}-R_d \\ 0 & \sqrt{(y-R_{34})^2+(z-R_d)^2}\ge t_{PLA}-R_d \end{cases}$$

On Developing a 3D-Printed Reservoir

Chapters 3 and 4 discuss two types of neuron connections in reservoirs: electrical and thermal coupling. Thermal coupling, as emphasized in Chapter 4, adds complexity to the reservoir's behaviour. To explore this further, we conducted an experiment using two types of reservoirs: one with only thermal coupling and the other with only electrical coupling among neurons. These setups are shown in Figure A.2. We created these reservoirs to see how different coupling mechanisms affect their performance. For the thermal coupling setup, we stacked five resistors on top of each other (Figure A.2 (a)), relying solely on thermal interactions between neurons. On the other hand, we printed a 2D structure with all neurons in one layer for the electrical coupling setup (Figure A.2 (b)), where information transmission happens through electrical interactions.



(a) (b)

Figure A.2. The 3D-printed reservoir consisting of (a) five thermally coupled neurons and (b) six electrically coupled neurons.

Comparing the performance of these reservoirs with different coupling mechanisms is shown in Figure A.3. One key observation is the impact of the number of neurons in the reservoir. Increasing the number of neurons improves the accuracy of processing time-series data. This implies that more neurons lead to better data modelling, enabling the reservoir to handle more complex tasks effectively. We found that thermally coupled neurons perform well in modelling time series data. This suggests that the thermal coupling mechanism adds analytical richness, making these neurons suitable for intricate tasks.

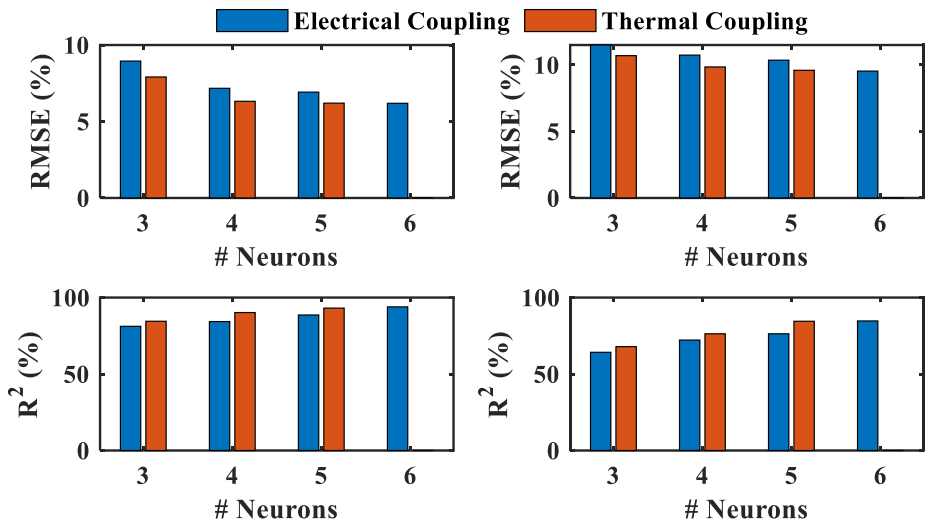


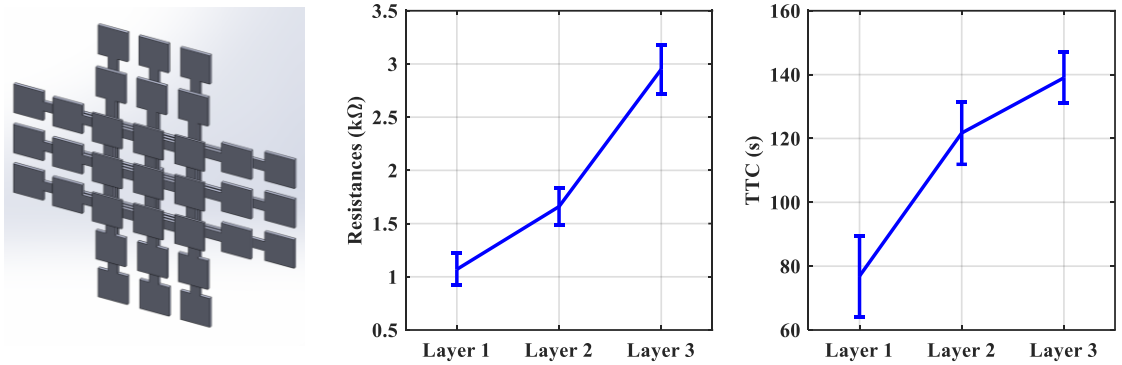
Figure A.3. Performance of the 3D-printed reservoir when employing different coupling mechanisms.

Additionally, comparing different coupling mechanisms, we noticed that a reservoir with six electrically coupled neurons can perform similarly to one with four thermally coupled neurons. This indicates that the choice of coupling mechanism directly influences the reservoir's efficiency and performance. Therefore, an intelligent approach involves utilizing a layered structure for thermal energy transfer to optimize reservoir efficiency. Doing so can reduce the number of neurons while maintaining performance.

We expanded our reservoir size to include 18 neurons by maintaining the same electrical coupling pattern across three layers while also incorporating thermal coupling (depicted in Figure A.4 (a)). This structured arrangement facilitates efficient heat transfer. Details concerning neuron resistance and thermal time constants are illustrated in Figure A.4 (b). The experimental results when solving different problems using the 3D-printed reservoir are shown in Figure A.5. When tackling more intricate challenges such as the NARMA10 task, the reservoir comprising 18 neurons distributed over three layers demonstrated superior performance compared to the reservoir with 12 neurons distributed over two layers. This heightened performance can be attributed to several factors, including the increased number of processing units and the added complexity introduced by thermal coupling.

In our exploration, we also delved into the effects of neuron widths on reservoir performance. A particularly intriguing approach emerged: the utilization of varying widths for neurons or resistors. This reservoir with random neuron widths was carefully crafted,

as Figure A.6 (a) depicts. Its distinctive characteristic lies in the diverse distribution of neuron resistances and thermal time constants across different layers, which is clearly illustrated in Figure A.6 (b). This unique arrangement aimed to fill the reservoir with heightened complexity, thereby fostering the potential for dynamic interactions.



(a) (b)

Figure A.4. (a) The 3D-printed uniform reservoir consisting of 3 thermally coupled layers (N = 18) and (b) distribution of the resistance and thermal time constants within the reservoir.

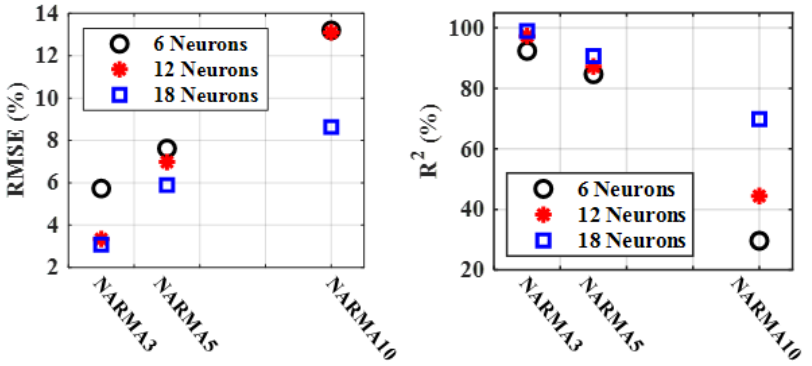
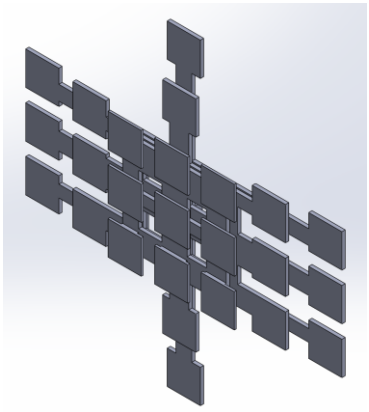


Figure A.5. Performance of the 3D-printed reservoir consisting of 3 thermally coupled layers (N = 18) with uniform width distribution for the resistors/neurons.

For a comprehensive assessment, we conducted a comparative analysis between the random 3D-printed reservoir and its uniform-width counterpart in Figure A.4. These reservoirs shared the same structure, consisting of three layers with 18 thermally coupled neurons, providing a fair basis for comparison. These reservoirs were subjected to solving various NARMA tasks, allowing us to evaluate their respective performance. The experimental results in Figure A.8 revealed a notable trend: the random network consistently exhibited superior accuracy when tackling various tasks. This enhanced

performance can be attributed to the inherent richness of dynamics in the random design. The variability in neuron widths introduces a layer of complexity that seems to enhance the reservoir's ability to handle a diverse range of tasks.

(a)



(b)

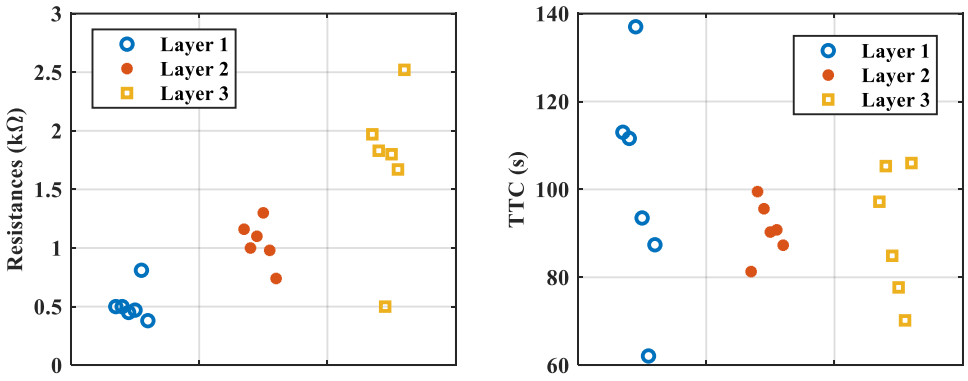


Figure A.6. **(a) The 3D-printed reservoir consisting of 3 thermally coupled layers (N = 18) and random width distribution, (b) the resistance distribution, and (c) thermal time constant distribution along different layers.**

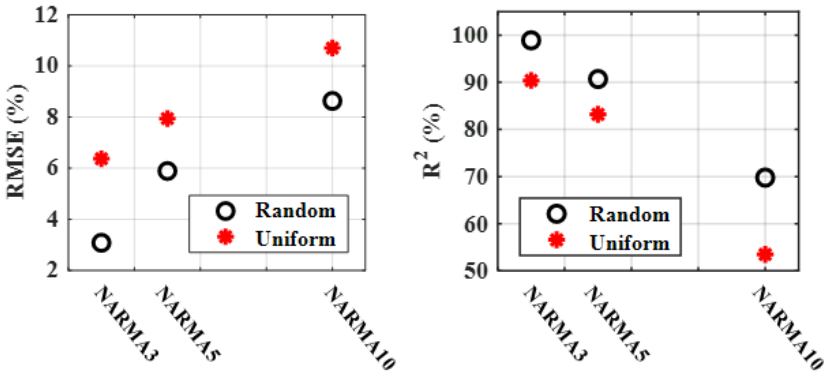


Figure A.7. **Performance of the 3D-printed reservoirs consisting of 3 thermally coupled layers (N = 18) with uniform and random width distribution for the resistors/neurons.**

Setting Up the Input Layer

In RC, the input layer serves a crucial role. It acts as the gateway for input signals, channelling them into the reservoir layer for processing. This layer is equipped with neurons that establish a connection with the incoming input, enabling the entire network to interact with external information. Before delving into the reservoir's processing tasks, it's imperative to ensure that the input signals are appropriately adjusted. This is similar to preparing a tool for a specific task – the tool needs to be appropriately set up for optimal performance. Similarly, for the reservoir to operate effectively, the input signals must be adjusted to match its requirements. This can involve standardizing the signals to have a consistent mean and variance or adapting them to a predefined range of values that align with the reservoir's architecture.

In the 3D-printed reservoir, there is a particular range of temperature at which each neuron can be safely driven to perform nonlinearly, showing thermal fading memory. The material's temperature-dependent resistivity changes the behaviour of the 3D-printed resistor both in terms of the temperature and the voltage across it. Figure A.8 shows this concept. The nonlinear behaviour starts at ~25-28^{o}C, and the conductive PLA melts at 50-60^{o}C. Thus, the temperature deviation of resistors should fall in ~ [7^{o}C 30^{o}C] in the presence of input signals. Hence, within our reservoir framework, it is imperative to maintain a controlled temperature environment for the neurons during input signal processing. Much like selecting the appropriate conditions for using a tool, we must ensure that the neurons are exposed to a temperature range that promotes their optimal function. This emphasis on temperature control is pivotal within our reservoir, as the neurons' behaviour hinges on these conditions.

The maximum input voltage signal to be mapped to a reservoir depends on the material properties (heat capacity, density, thermal conductivity, and electrical resistivity/conductivity) and the reservoir itself (the number of thermally coupled layers, dimensions of the printed resistors, and the gap between the resistors). Scaling the input signal, a, can help improve the reservoir's performance by ensuring that the input values fall within a range that the network can effectively process. Suppose the input values are too large or too small. In that case, this can lead to problems with melting the entire reservoir or losing the reservoir's computational power due to tapping into the linear region

of the nonlinear dynamic system. The ideal input scaling a is achieved when the environment temperature is set to room temperature.

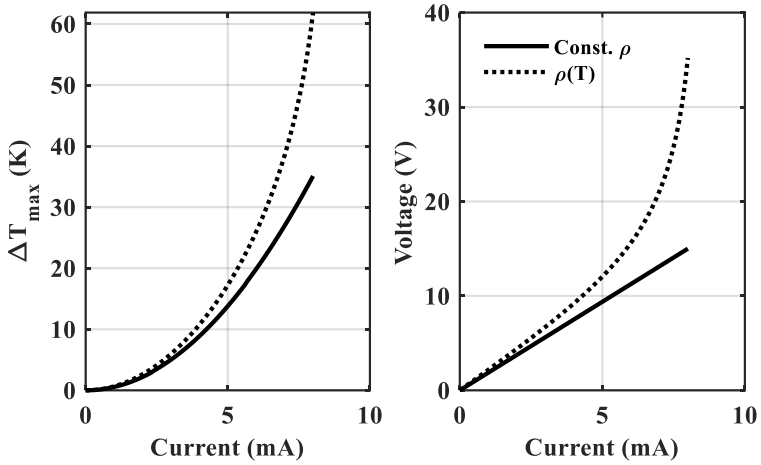


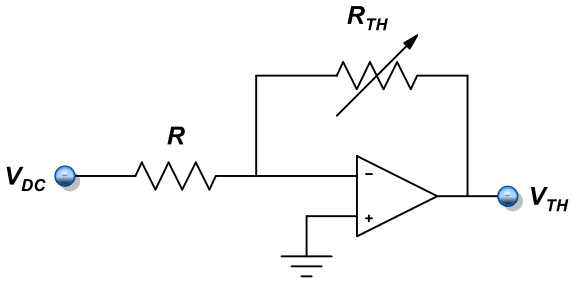
Figure A.8. The nonlinearity of the material depends on the temperature of the resistor.

The performance of the reservoir was studied under different environmental temperatures, denoted as T_{env}. The reservoir was placed inside an oven, as shown in A.9 (a) and the temperature using the temperature control circuitry. In these experiments, we adjusted the input scaling of the 3D-printed RC and retrained the output layer to observe the effects of varying environmental temperatures on the system's performance. We set up a controlled experimental environment to investigate the impact of environmental temperature on the system's behaviour. The 3D-printed reservoir was placed inside an oven to manipulate the environmental temperature. This allowed simulations of different temperature conditions to observe how the system responded. We first adjusted the environmental temperature to a specific value in each experiment, creating a controlled setting. To monitor T_{env}, the circuit in Figure A.9 (b) was designed using an NTC thermistor (NT0515291 from Ametherm), in which

$$V_{TH} = -\frac{R_{TH}}{\mathrm{R}}\, V_{DC} = -\frac{R_a}{\mathrm{R}} e^{\beta\left(\frac{1}{T_{env}} - \frac{1}{T_a}\right)} V_{DC} \qquad \text{(A-7)}$$

If we set the $R = R_a$ and $V_{DC} = 1V$, the output voltage from this circuit would fall in the range of $[-1\ \ 0]V$. Therefore, the environment temperature is given as

(a)



(b)

Figure A.9. (a) The experimental setup for studying the effect of environmental temperature and (b) Environment temperature monitoring circuitry used in the experimental temperature studies.

$$T_{env} = \frac{1}{\frac{\ln(|V_{TH}|)}{\beta} + \frac{1}{T_a}} \tag{A-8}$$

We then modified the input scaling parameter to account for the influence of temperature on input signals, considering $T \propto \frac{V^2}{R}$ for each neuron:

$$a_{adjust} = a\sqrt{\frac{(T_{max} - T_{env})}{(T_{max} - T_a)}} \tag{A-9}$$

This step was crucial to ensure that the system adapted to the changing environmental conditions, maintaining its efficiency in processing. Following the input scaling adjustment, the system's output layer was retrained based on the modified input signals. This retraining process involved updating the output weights to align with the newly scaled input signals, effectively tuning the system for optimal performance under the specific environmental temperature. We repeated this process for various temperature settings to create a comprehensive dataset of system behaviours under different

environmental conditions. The outcomes of each experiment were then recorded and summarized in Table A.4. The results provided valuable information on how the system's performance evolved across different temperatures, demonstrating the necessity of adapting input scaling to maintain efficient operation across varying environmental conditions in thermal reservoir computing systems. Therefore, to make the reservoir adaptable across varying environmental temperatures, modifications in input scaling are necessary, which in turn necessitates retraining the reservoir on training data. This strategic approach ensures that the reservoir can effectively accommodate diverse operating conditions, showcasing the critical interplay between environmental factors and input scaling in the pursuit of optimized thermal reservoir computing platforms.

Table A.4. 3D-printed reservoir's performance under different environment temperatures with adjusted scaling.

Temperature (°C)	*Statistical Evaluation Task*					
	NARMA3		*NARMA5*		*NARMA10*	
	RMSE (%)	*R^2 (%)*	*RMSE (%)*	*R^2 (%)*	*RMSE (%)*	*R^2 (%)*
22	3.45	97.32	6.8	87.8	10.4	70.2
28	3.88	95.5	4.99	93.2	11.9	67.6
32	4.39	93.4	5.99	90.2	10.5	73.3
38	4.91	91.3	6.9	88.4	9.98	74.64

Design of the Output Layer

The output layer can be trained by running a linear regression on all the outputs from the reservoir in MATLAB®. Once trained, it can be replaced with an operational amplifier (OpAmp) circuit, as shown in Figure A.10, in which the ratio of the resistors in the feedback and input determine the trained weights. Let us consider we have ℓ trainable weights, of which *q* negative weights are the result after training. The output signal would take the form of

$$V_{out} = \frac{1 + \sum_{i=1}^{q} \frac{R_f}{R_i}}{1 + \sum_{i=q+1}^{\ell} \frac{R_f}{R_i}} \sum_{i=q+1}^{\ell} \frac{R_f}{R_i} V_i - \sum_{i=1}^{q} \frac{R_f}{R_i} V_i \tag{A-10}$$

where V_i is the signals received from the reservoir. To prevent the loading effects on the reservoir, it is safe to consider $R_f = 1M\Omega$, the rest of the resistor values can be derived based on:

$$W_{out}^{(\ell\times 1)} = \begin{cases} -\dfrac{R_f}{R_i} & i \le q \quad (negative\ weigths) \\ \dfrac{1 + \sum_{i=1}^{q} \dfrac{R_f}{R_i}}{1 + \sum_{i=q+1}^{\ell} \dfrac{R_f}{R_i}} \dfrac{R_f}{R_i} & i > q \quad (positive\ weigths) \end{cases} \tag{A-11}$$

We solved the equation above in Maple for the proposed 3D-printed processor once trained for solving a specific problem.

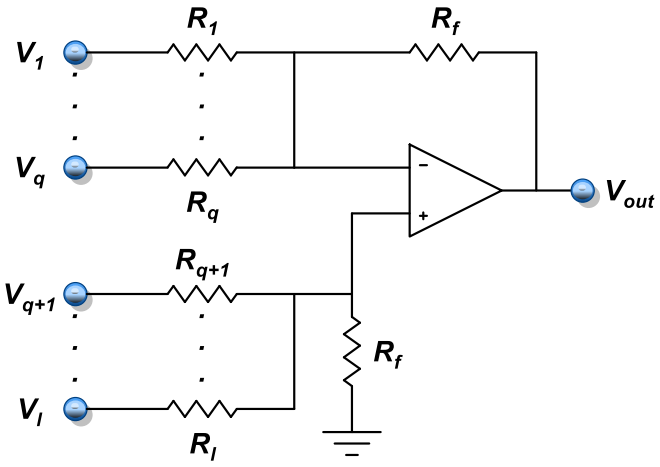


Figure A.11. The output layer of the proposed computing platform.

Proposed Design Procedure and Rules

Developing a physical reservoir computing platform involves a systematic approach that draws upon insights gained from practical experiences. Through an exploration of various design aspects, from selecting appropriate materials to configuring coupling mechanisms, a general procedure emerges that guides the creation of efficient and adaptable computational systems. Rooted in my own experiential journey, this design process intertwines multiple realms of science, culminating in creating reservoirs that harness the nuances of physical properties for information processing. This narrative

encapsulates the steps and considerations that pave the way for the development of diverse and potent physical reservoir computing platforms.

- Identify the physical system: Determine the physical system to use as the reservoir. It can be any physical system that exhibits rich dynamical behaviour, such as a chaotic electrical circuit, a fluid flow system, or a mechanical system. Make sure there is memory and nonlinearity in the system's response to physical stimulation.

- Measure the response time of the physical dynamical system to know the time scale and the speed of the resulting processor.

- Choose the sensing method: Select an appropriate sensing method for measuring the state variables of the physical system. It can involve using sensors, transducers, or other measurement devices.

- Look for appropriate energy transfer mechanisms to implement in the dynamical system for information flow among the neurons. It will ensure the presence of nonlinear interactions among the neurons.

- Design the input layer of the physical reservoir computing platform, which should be capable of converting the input signals into a suitable range (i.e., operating point) that can be injected into the physical system.

- Construct the physical reservoir by connecting the sensing system to the input and output layers. The reservoir should be designed to exhibit complex and diverse dynamical behaviour.

- Train the output layer of the physical reservoir computing platform using suitable machine learning algorithms. It involves using linear or nonlinear regression techniques or other machine learning algorithms.

- Optimize the physical reservoir computing platform by tuning the various parameters, such as the connectivity of the reservoir, to achieve the best possible performance.

- Test and evaluate the physical reservoir computing platform using appropriate benchmarks and datasets. It can involve using standard time-series prediction tasks, such as the NARMA task, delayed XOR, or other real-world datasets.

- Refine and improve the physical reservoir computing platform based on the evaluation results, and continue to optimize and refine the platform as necessary to achieve the best possible performance.

Appendix B.

MATLAB Codes

The Recurrent Neural Networks

```
%%%% The layered RNN, or Layrecnet, is a recurrent neural network architecture
% implemented in MATLAB® that is used for sequence-to-sequence prediction tasks.
% This architecture includes multiple recurrent layers that are stacked on top
% of each other, with each layer taking the output of the previous layer as input.
% The Layrecnet architecture is particularly useful for tasks that require modeling
% complex sequences

clc
close all
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% Initialization %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

RealVal  = load('RealValue_N3.csv');
RealVal  = RealVal/max(RealVal);

data  = load('input2_310V.txt');
In0 = data(:,2)/max(data(:,2));
In1 = circshift(In0,1);
In2 = circshift(In0,2);
In3 = circshift(In0,3);
In = [In0 In1 In2 In3];

N1 = length(In0);
N2 = length(RealVal);
N = min(N1,N2);

%%%%%%% Train/Test Segmentation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n = 10;
N = N - n;
Ratio = 0.7;
NTrain = ceil(Ratio*N);

trainData_In = In(1:NTrain,:);
trainData_RealVal = RealVal(1:NTrain,:);
testData_In = In(NTrain:N,:);
testData_RealVal = RealVal(NTrain:N,:);

%%%%%%% Create the Recurrent Neural Network %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define the number of input, hidden, and output nodes
input_nodes = 4;
hidden_nodes = [8 8 8];
output_nodes = 1;
layerDelays = [1 2 3];

% Define the activation functions for the hidden layers
hiddenActivationFunctions = {'logsig', 'logsig', 'logsig'};
outputActivationFunction = 'purelin';

% Define the recurrent neural network
net = layrecnet(layerDelays, hidden_nodes, 'traingdx');

% Set the nonlinear functions for the hidden layers and the output layer
net.layers{1}.transferFcn = hiddenActivationFunctions{1};
net.layers{2}.transferFcn = hiddenActivationFunctions{2};
net.layers{3}.transferFcn = hiddenActivationFunctions{3};
net.layers{4}.transferFcn = outputActivationFunction;
```

```
% Define the input and output data for training
input_data = trainData_In;
target_data = trainData_RealVal;

for i = 1:50
% Train the recurrent neural network
net = train(net, input_data', target_data');
view(net)
% nnet.guis.closeAllViews()



% Use the trained network to predict the output for the test data
y_pred = net(testData_In');

% Calculate the RMSE and R-squared value
rmse(i) = sqrt(mean((y_pred - testData_RealVal').^2));
r_squared(i) = 1 - (sum((y_pred - testData_RealVal').^2) / sum((testData_RealVal' -
mean(testData_RealVal')).^2));

end

rmse = sum(rmse)/50;
r_squared = sum(r_squared)/50;

% Print the results
fprintf('RMSE: %.4f\n', rmse);
fprintf('R-squared: %.4f\n', r_squared);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure
axes4 = axes('Parent',fig,'FontSize',13,'FontName','Times New
Roman','FontWeight','bold','LineWidth',1);
box(axes4,'on');
hold(axes4,'on');
grid on
%xlim([0 250])
%ylim([5.5 8.5])

plot(testData_RealVal,'b','LineWidth',2); hold on
plot(y_pred,'r','LineWidth',2); hold on

xlabel('Timestep','FontWeight','bold','FontSize',14);
ylabel('Norm. Output','FontWeight','bold','FontSize',14);
legend({'Real Values','Predicted Values'},'Location','southeast','FontName','Times New
Roman','FontWeight','bold','FontSize',12)
legend('boxoff')

set(fig, 'Position', [300, 50, 600, 200]);
```

The Feedforward Neural Network

```
%%%%% The feedforward neural network (FNN) is a type of artificial neural
% network that is widely used in machine learning and deep learning applications.
% In MATLAB®, the FNN can be created using the feedforwardnet function, which
% constructs a multilayer perceptron (MLP) architecture. The feedforwardnet
% function allows the user to specify the number of hidden layers, the number
% of neurons in each layer, the activation functions used in each layer, and other
% parameters.

clc
close all
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% Initialization %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

RealVal  = load('RealValue_N1.csv');
RealVal  = RealVal/max(RealVal);
data  = load('input2_310V.txt');
In0 = data(:,2)/max(data(:,2));
In1 = circshift(In0,1);
In2 = circshift(In0,2);
In3 = circshift(In0,3);
In = [In0 In1 In2 In3];

N1 = length(In0);
N2 = length(RealVal);
N = min(N1,N2);

%%%%%%% Train/Test Segmentation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n = 10;
N = N - n;
Ratio = 0.7;
NTrain = ceil(Ratio*N);

trainData_In = In(1:NTrain,:);
trainData_RealVal = RealVal(1:NTrain,:);
testData_In = In(NTrain:N,:);
testData_RealVal = RealVal(NTrain:N,:);

%%%%%%% Create the Feedforward Neural Network %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define the number of inputs, hidden layers, nodes and outputs
inputs = 4;
hiddenLayers = 3;
% hiddenNodes = 18;
hiddenNodes = [6 200 10];
outputs = 1;

% Define the nonlinear functions for the hidden layers and the output layer
hiddenActivationFunctions = {'logsig', 'logsig', 'logsig'};
outputActivationFunction = 'purelin';

% 'purelin' for a linear transfer function
% 'logsig' for a log-sigmoid transfer function
% 'tansig' for a hyperbolic tangent sigmoid transfer function
% 'radbas' for a radial basis transfer function
% 'hardlim' for a hard limit transfer function

for i = 1:50
% Create the neural network with the defined architecture
net = feedforwardnet(hiddenNodes, 'traingdx');

% Set the nonlinear functions for the hidden layers and the output layer
net.layers{1}.transferFcn = hiddenActivationFunctions{1};
net.layers{2}.transferFcn = hiddenActivationFunctions{2};
net.layers{3}.transferFcn = hiddenActivationFunctions{3};
net.layers{2}.transferFcn = outputActivationFunction;

% Configure the network for training
net = configure(net, In', RealVal');
view(net)
% nnet.guis.closeAllViews()

% Train the network using the input and output data
```

```
[net, tr] = train(net, trainData_In', trainData_RealVal');

% Use the trained network to predict the output for the test data
y_pred = net(testData_In');

% Calculate the RMSE and R-squared value
rmse(i) = sqrt(mean((y_pred - testData_RealVal').^2));
r_squared(i) = 1 - (sum((y_pred - testData_RealVal').^2) / sum((testData_RealVal' -
mean(testData_RealVal')).^2));

end

rmse = sum(rmse)/50;
r_squared = sum(r_squared)/50;

% Print the results
fprintf('RMSE: %.4f\n', rmse);
fprintf('R-squared: %.4f\n', r_squared);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure
axes4 = axes('Parent',fig,'FontSize',13,'FontName','Times New
Roman','FontWeight','bold','LineWidth',1);
box(axes4,'on');
hold(axes4,'on');
grid on
%xlim([0 250])
%ylim([5.5 8.5])

plot(testData_RealVal,'b','LineWidth',2); hold on
plot(y_pred,'r','LineWidth',2); hold on

xlabel('Timestep','FontWeight','bold','FontSize',14);
ylabel('Norm. Output','FontWeight','bold','FontSize',14);
legend({'Real Values','Predicted Values'},'Location','southeast','FontName','Times New
Roman','FontWeight','bold','FontSize',12)
legend('boxoff')

set(fig, 'Position', [300, 50, 600, 200]);
```

Appendix C.

Maple Codes

Deriving Static Solutions for Resistor 1

```
#Thermal Analysis Steady State, Constant rho, Resistors in Layer 1

restart;


EQ1 := 1/r*diff(k*r*diff(T1a(r),r),r) + e_gen1 = 0;

EQ2 := 1/r*diff(k_PLA*r*diff(T1b(r),r),r) = 0;


A1 := Pi/2*r1^2:                A_hflux1 := Pi*r1*L:                q_gen1 := e_gen1 * L * A1 / (A_hflux1);


BC1 := eval(-k_PLA*diff(T1b(r),r),r=r1)= eval(-k*diff(T1a(r),r),r=r1) + q_gen1;

BC2 := T1b(t_PLA)=0;

BC3 := eval(T1a(r),r=r1) = eval(T1b(r),r=r1);

BC4 := eval(diff(T1a(r),r),r=0) = 0;


SOL := dsolve({EQ1,EQ2,BC1,BC2,BC3,BC4},{T1a(r),T1b(r)}):

T1a(r) := rhs(SOL[1]);                T1b(r) := rhs(SOL[2]);

q1 := e_gen1 + eval(-k*diff(T1a(r),r),r=r1);  q:= eval(-k_PLA*diff(T1b(r),r),r=t_PLA);


T1(r) := piecewise(r<=r1,T1a(r),r<t_PLA,T1b(r),r>t_PLA,0);

T1(y,z) := eval(T1(r),r=sqrt(z^2+y^2));


sigma:=3.3:     rho:=1/sigma:     L:=100e-3:     r0:=5e-6:     t_PLA:=10e-3:     Iin:=10e-6:
Ta:=293.15:     k:=0.2:           k_PLA:=0.11:   L_PLA:=100e-3:  W_PLA:=100e-3:  rho_m:=1210:
Cp:=1180:       r1:=sqrt(2)*r0;

A1 := Pi/2*r1^2;          R1 := rho*L/A1;

E_gen1 := R1*Iin^2;       e_gen1 := E_gen1/(L*A1);          q1 := e_gen1 * r1;


plot(T1(r),r=0...t_PLA);

plot3d(T1(y,z), y = -t_PLA...t_PLA, z = r0..t_PLA);
```

The Maple solution for Resistor 1

$$EQ1 := \frac{k\left(\frac{d}{dr}\,T1a(r)\right) + k\,r\left(\frac{d^2}{dr^2}\,T1a(r)\right)}{r} + e\_gen1 = 0$$

$$EQ2 := \frac{k\_PLA\left(\frac{d}{dr}\,T1b(r)\right) + k\_PLA\,r\left(\frac{d^2}{dr^2}\,T1b(r)\right)}{r} = 0$$

$$BC1 := -k\_PLA\left(\frac{d}{dr1}\,T1b(r1)\right) = -k\left(\frac{d}{dr1}\,T1a(r1)\right) + \frac{e\_gen1\,r1}{2}$$

$$BC2 := T1b(t\_PLA) = 0$$

$$BC3 := T1a(r1) = T1b(r1)$$

$$BC4 := \left(\frac{d}{dr}\,T1a(r)\right)\Bigg|_{\{r=0\}} = 0$$

$$T1a(r) := -\frac{e\_gen1\,r^2}{4\,k} + \frac{e\_gen1\,r1^2\,(4\ln(t\_PLA)\,k - 4\ln(r1)\,k + k\_PLA)}{4\,k\,k\_PLA}$$

$$T1b(r) := -\frac{e\_gen1\,r1^2\ln(r)}{k\_PLA} + \frac{e\_gen1\,r1^2\ln(t\_PLA)}{k\_PLA}$$

$$q1 := e\_gen1 + \frac{1}{2}\,e\_gen1\,r1$$

$$q := \frac{e\_gen1\,r1^2}{t\_PLA}$$

$$T1(r) := \begin{cases} -\frac{e\_gen1\,r^2}{4\,k} + \frac{e\_gen1\,r1^2\,(4\ln(t\_PLA)\,k - 4\ln(r1)\,k + k\_PLA)}{4\,k\,k\_PLA} & r \le r1 \\ -\frac{e\_gen1\,r1^2\ln(r)}{k\_PLA} + \frac{e\_gen1\,r1^2\ln(t\_PLA)}{k\_PLA} & r < t\_PLA \\ 0 & t\_PLA < r \end{cases}$$

$$T1(y,z) := \begin{cases} -\frac{e\_gen1\left(y^2+z^2\right)}{4\,k} + \frac{e\_gen1\,r1^2\,(4\ln(t\_PLA)\,k - 4\ln(r1)\,k + k\_PLA)}{4\,k\,k\_PLA} & \sqrt{y^2+z^2} \le r1 \\ -\frac{e\_gen1\,r1^2\ln\left(y^2+z^2\right)}{2\,k\_PLA} + \frac{e\_gen1\,r1^2\ln(t\_PLA)}{k\_PLA} & \sqrt{y^2+z^2} < t\_PLA \\ 0 & t\_PLA < \sqrt{y^2+z^2} \end{cases}$$

Deriving Static Solutions for Resistor 3

```
#Thermal Analysis Steady State, Constant rho, Resistors in Layer 2

restart;


EQ1 := 1/r*diff(k*r*diff(T3a(r),r),r) + e_gen3 = 0;

EQ2 := 1/r*diff(k_PLA*r*diff(T3b(r),r),r) = 0;


A3 := Pi*r3^2:          A_hflux3 := 2*Pi*r3*L:          q_gen3 := e_gen3 * L * A3 / (A_hflux3);


BC1 := eval(-k_PLA*diff(T3b(r),r),r=r3)= eval(-k*diff(T3a(r),r),r=r3) + q_gen3;

BC2 := T3b(t_PLA-R13)=0;                BC2_p := T3b(t_PLA+R13)=0;

BC3 := eval(T3a(r),r=r3) = eval(T3b(r),r=r3);

BC4 := eval(diff(T3a(r),r),r=0) = 0;


SOL :=     dsolve({EQ1,EQ2,BC1,BC2,BC3,BC4},{T3a(r),T3b(r)}):

SOL_p := dsolve({EQ1,EQ2,BC1,BC2_p,BC3,BC4},{T3a(r),T3b(r)}):

T3a(r) := rhs(SOL[1]);   T3a_p(r) := rhs(SOL_p[1]);

T3b(r) := rhs(SOL[2]);   T3b_p(r) := rhs(SOL_p[2]);


q3 := q_gen3 + eval(-k*diff(T3a(r),r),r=r3);


T3(r) := piecewise(r<=r3,T3a(r),r<t_PLA-R13,T3b(r),r>t_PLA-R13,0);

T3_p(r) := piecewise(r<=r3,T3a_p(r),r<t_PLA+R13,T3b_p(r),r>t_PLA+R13,0);

T3(y,z) := eval(T3(r),r=sqrt((z-R13)^2+y^2))+eval(T3_p(r),r=sqrt((z+R13)^2+y^2));


sigma:=3.3:        rho:=1/sigma:      L:=100e-3:         r0:=5e-6:          t_PLA:=10e-3:      Iin:=10e-6:
Ta:=293.15:        k:=0.2:            k_PLA:=0.11:       R13:=2e-3:         L_PLA:=100e-3:     W_PLA:=100e-3:
rho_m:=1210:       Cp:=1180:          r3:=r0;

A3 := Pi*r3^2;                  R3 := rho*L/A3;

E_gen3 := R3*Iin^2;             e_gen3:= E_gen3/(A3*L);                 q3 := e_gen3 * r3;


plot(T3(r),r=0...t_PLA); plot(eval(T3(y,z),y=r1/sqrt(2)),z=0...t_PLA);

plot3d(T3(y,z), y = -t_PLA...t_PLA, z = r0..t_PLA);
```

The Maple solution for Resistor 3

$$EQ1 := \frac{k\left(\frac{d}{dr}\,T3a(r)\right) + k\,r\left(\frac{d^2}{dr^2}\,T3a(r)\right)}{r} + e\_gen3 = 0$$

$$EQ2 := \frac{k\_PLA\left(\frac{d}{dr}\,T3b(r)\right) + k\_PLA\,r\left(\frac{d^2}{dr^2}\,T3b(r)\right)}{r} = 0$$

$$q\_gen3 := \frac{e\_gen3\,r3}{2}$$

$$BC1 := -k\_PLA\left(\frac{d}{dr3}\,T3b(r3)\right) = -k\left(\frac{d}{dr3}\,T3a(r3)\right) + \frac{e\_gen3\,r3}{2}$$

$$BC2 := T3b(t\_PLA - R13) = 0$$

$$BC2\_p := T3b(t\_PLA + R13) = 0$$

$$BC3 := T3a(r3) = T3b(r3)$$

$$BC4 := \left(\frac{d}{dr}\,T3a(r)\right)\Bigg|_{\{r=0\}} = 0$$

$$T3a(r) := -\frac{e\_gen3\,r^2}{4\,k} + \frac{e\_gen3\,r3^2\,(4\ln(t\_PLA - R13)\,k - 4\ln(r3)\,k + k\_PLA)}{4\,k\,k\_PLA}$$

$$T3a\_p(r) := -\frac{e\_gen3\,r^2}{4\,k} + \frac{e\_gen3\,r3^2\,(4\ln(t\_PLA + R13)\,k - 4\ln(r3)\,k + k\_PLA)}{4\,k\,k\_PLA}$$

$$T3b(r) := -\frac{e\_gen3\,r3^2\ln(r)}{k\_PLA} + \frac{e\_gen3\,r3^2\ln(t\_PLA - R13)}{k\_PLA}$$

$$T3b\_p(r) := -\frac{e\_gen3\,r3^2\ln(r)}{k\_PLA} + \frac{e\_gen3\,r3^2\ln(t\_PLA + R13)}{k\_PLA}$$

$$T3(r) := \begin{cases} -\frac{e\_gen3\,r^2}{4\,k} + \frac{e\_gen3\,r3^2\,(4\ln(t\_PLA - R13)\,k - 4\ln(r3)\,k + k\_PLA)}{4\,k\,k\_PLA} & r \le r3 \\ -\frac{e\_gen3\,r3^2\ln(r)}{k\_PLA} + \frac{e\_gen3\,r3^2\ln(t\_PLA - R13)}{k\_PLA} & r < t\_PLA - R13 \\ 0 & t\_PLA - R13 < r \end{cases}$$

$$T3\_p(r) := \begin{cases} -\frac{e\_gen3\,r^2}{4\,k} + \frac{e\_gen3\,r3^2\,(4\ln(t\_PLA + R13)\,k - 4\ln(r3)\,k + k\_PLA)}{4\,k\,k\_PLA} & r \le r3 \\ -\frac{e\_gen3\,r3^2\ln(r)}{k\_PLA} + \frac{e\_gen3\,r3^2\ln(t\_PLA + R13)}{k\_PLA} & r < t\_PLA + R13 \\ 0 & t\_PLA + R13 < r \end{cases}$$

$$T3(y,z) := \begin{cases} -\frac{e\_gen3\left((z - R13)^2 + y^2\right)}{4\,k} + \frac{e\_gen3\,r3^2\,(4\ln(t\_PLA - R13)\,k - 4\ln(r3)\,k + k\_PLA)}{4\,k\,k\_PLA} & \sqrt{(z - R13)^2 + y^2} \le r3 \\ -\frac{e\_gen3\,r3^2\ln\left((z - R13)^2 + y^2\right)}{2\,k\_PLA} + \frac{e\_gen3\,r3^2\ln(t\_PLA - R13)}{k\_PLA} & \sqrt{(z - R13)^2 + y^2} < t\_PLA - R13 \\ 0 & t\_PLA - R13 < \sqrt{(z - R13)^2 + y^2} \end{cases}$$

$$+ \begin{cases} -\frac{e\_gen3\left((z + R13)^2 + y^2\right)}{4\,k} + \frac{e\_gen3\,r3^2\,(4\ln(t\_PLA + R13)\,k - 4\ln(r3)\,k + k\_PLA)}{4\,k\,k\_PLA} & \sqrt{(z + R13)^2 + y^2} \le r3 \\ -\frac{e\_gen3\,r3^2\ln\left((z + R13)^2 + y^2\right)}{2\,k\_PLA} + \frac{e\_gen3\,r3^2\ln(t\_PLA + R13)}{k\_PLA} & \sqrt{(z + R13)^2 + y^2} < t\_PLA + R13 \\ 0 & t\_PLA + R13 < \sqrt{(z + R13)^2 + y^2} \end{cases}$$

Plot Static Solutions for all Resistors 1, 2, 3, and 4

#Thermal Analysis Steady State, Constant rho

restart;

T1(y, z) := piecewise(sqrt(y^2 + z^2) <= r1, -e_gen1*(y^2 + z^2)/(4*k) - e_gen1*r1^2*(4*ln(r1)*k - 4*ln(t_PLA)*k - k_PLA)/(4*k*k_PLA), sqrt(y^2 + z^2) < t_PLA, -e_gen1*r1^2*ln(y^2 + z^2)/(2*k_PLA) + e_gen1*r1^2*ln(t_PLA)/k_PLA, t_PLA < sqrt(y^2 + z^2), 0);

T3(y, z) := piecewise(sqrt((z - R13)^2 + y^2) <= r3, -e_gen3*((z - R13)^2 + y^2)/(4*k) + e_gen3*r3^2*(4*ln(t_PLA - R13)*k - 4*ln(r3)*k + k_PLA)/(4*k*k_PLA), sqrt((z - R13)^2 + y^2) < t_PLA - R13, -e_gen3*r3^2*ln((z - R13)^2 + y^2)/(2*k_PLA) + e_gen3*r3^2*ln(t_PLA - R13)/k_PLA, t_PLA - R13 < sqrt((z - R13)^2 + y^2), 0) + piecewise(sqrt((z + R13)^2 + y^2) <= r3, -e_gen3*((z + R13)^2 + y^2)/(4*k) + e_gen3*r3^2*(4*ln(t_PLA + R13)*k - 4*ln(r3)*k + k_PLA)/(4*k*k_PLA), sqrt((z + R13)^2 + y^2) < t_PLA + R13, -e_gen3*r3^2*ln((z + R13)^2 + y^2)/(2*k_PLA) + e_gen3*r3^2*ln(t_PLA + R13)/k_PLA, t_PLA + R13 < sqrt((z + R13)^2 + y^2), 0);

T2(y,z) := eval(T1(y,z),y=y-R12);

T4(y,z) := eval(T3(y,z),y=y-R34);

T12(y,z) := T1(y,z)+ T2(y,z);

T13(y,z) := T1(y,z)+ T3(y,z);

T14(y,z) := T1(y,z)+ T4(y,z);

T23(y,z) := T2(y,z)+ T3(y,z);

T24(y,z) := T2(y,z)+ T4(y,z);

T34(y,z) := T3(y,z)+ T4(y,z);

T(y,z) := T1(y,z)+ T2(y,z)+T3(y,z)+ T4(y,z);

sigma:=3.3:	rho:=1/sigma:	L:=100e-3:	r0:=5e-6:	t_PLA:=10e-3:	Iin:=10e-6:
Ta:=293.15:	k:=0.2:	k_PLA:=0.11:	L_PLA:=100e-3:	W_PLA:=100e-3:	rho_m:=1210:
Cp:=1180:	r1:=sqrt(2)*r0:	r3:=r0:	r2:=r1:	r4:=r3:	R13:=2e-3:
R12:=R13:	R34:=R13:				

A1 := Pi/2*r1^2;

A3 := Pi*r3^2;

R1 := rho*L/A1;

R3 := rho*L/A3;

E_gen1 := R1*Iin^2;

E_gen3 := R3*Iin^2;

e_gen1 := E_gen1/(L*A1);

e_gen3 := E_gen3/(L*A3);

plot3d(T12(y,z), y = -t_PLA...t_PLA, z = sqrt(2)*r0..t_PLA);

plot3d(T13(y,z), y = -t_PLA...t_PLA, z = sqrt(2)*r0..t_PLA);

plot3d(T14(y,z), y = -t_PLA...t_PLA, z = sqrt(2)*r0..t_PLA);

plot3d(T23(y,z), y = -t_PLA...t_PLA, z = sqrt(2)*r0..t_PLA);

plot3d(T24(y,z), y = -t_PLA...t_PLA, z = sqrt(2)*r0..t_PLA);

plot3d(T34(y,z), y = -t_PLA...t_PLA, z = sqrt(2)*r0..t_PLA);

plot3d(T(y,z), y = -t_PLA...t_PLA, z = sqrt(2)*r0..t_PLA);