A Novel Machine Learning Model Enabled by Perovskite-Memristor-based Hardware

Ivan Shmarov

Thesis submitted for the degree of

Doctor of Philosophy

School of Mathematics, Statistics & Physics

Newcastle University

Newcastle upon Tyne

United Kingdom

December 20, 2023

To my loving family.

Acknowledgements

I would like to express my gratitude to my supervisors, Dr Pablo Docampo, Dr Thomas Billam, and Dr Rishad Shafik, for their unwavering support and patience throughout my PhD journey. Also, I am immensely grateful to the fellow students I have met along the way who have provided me with companionship. Additionally, I am grateful to my ever-loving family for their continuous support and encouragement.

Abstract

Machine learning (ML) offers the automation of routine yet highly complex decision-making processes, previously regarded as only a human job. However, ML is still limited by the amount of computational resources it requires to operate. As such, employing machine learning for decision-making on the spot, where the input data originates, is challenging due to resource constraints associated with such settings. In this thesis, I present my contribution to the solution of this problem in the form of a novel ML algorithm, Fuzlearn, a model of a new electrical component, namely memristor, that can be utilised for efficient hardware acceleration of Fuzlearn, and, finally, a circuit simulation of the said hardware accelerator.

In the first part of the thesis, I review the drawbacks of already existing ML algorithms, namely neural networks (NNs) and Tsetlin machines (TMs), which limit their applicability for in-situ ML. Additionally, I explore the potential of memristors for ML hardware implementation/acceleration, with Halide Perovskites emerging as the most promising candidate due to Perovskites' inherent memristive nature and ease of fabrication.

In Chapter 3, I introduce Fuzlearn - a novel ML algorithm, based on a Tsetlin machine architecture. I present its working mechanism, which performs analogue-to-Boolean classification by learning appropriate input class boundaries. I demonstrate its effectiveness through extensive testing on various real-world ML problems, highlighting its capability to process analogue inputs effectively, as well as demonstrating potential caveats that can be encountered while using Fuzlearn.

In Chapter 4, I delve into the development of Perovskite memristors, presenting and comparing different device configurations for optimal usage within ML hardware. Through rigorous testing, I confirm that filament-formation silver-electrode design yields the most promising memristive properties. As a major contribution to this chapter, I created and verified a simplistic yet powerful model of Perovskite memristors suitable for rapid circuit design applications.

Finally, Chapter 5 focuses on integrating Fuzlearn into hardware by leveraging my Perovskite memristor model. I present my circuit design, which successfully demonstrates the algorithm's functionality within circuit simulation software, proving its potential for practical implementation as a hardware-accelerated architecture for machine learning tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A dream of in-situ Machine Learning

Machine learning (ML) possesses the remarkable ability to automate complex decision-making processes, significantly benefiting various aspects of our lives [1]. By leveraging advanced algorithms and large datasets, ML systems can analyse intricate patterns and make informed choices, sparing us from the burdensome task of manual decision-making. Through the automation of intricate decision-making, ML empowers us to focus on higher-level tasks and fosters efficiency, accuracy, and innovation across numerous domains.

The concept of conducting machine learning at the source of data/input, i.e. in-situ ML, holds immense promise and brings numerous advantages [2]. The traditional approach to ML involves transferring data to a central processing location for analysis, which can be time-consuming [3], resource-intensive [4], and potentially compromising data security and privacy [5]. In contrast, performing ML in situ eliminates the need for data movement, enabling real-time analysis and decision-making at the source. This approach not only saves valuable time but also reduces network bandwidth requirements. By processing data where it originates, whether it's within the framework of the Internet of Things (IoT) [6, 7], edge servers [8], or other distributed systems, in-situ ML unlocks the potential for rapid insights, more efficient energy distribution, and improved data governance.

Implementing in-situ ML poses significant challenges primarily due to the lack of ap-

propriate architectures for the task. Neural Networks (NNs), being the most common choice of ML architecture [9], are ill-suited for such undertaking because of their inherent limitations. NNs are run on conventional computing hardware, which requires a constant power supply, making it challenging to deploy them in resource-constrained environments where power availability may be limited or intermittent [10]. Additionally, in-situ ML implies processing data from various sensors or analogue sources, necessitating analogue-to-digital conversion, which can introduce noise/precision issues and additional energy consumption [11]. Furthermore, NNs are often criticized for being opaque black boxes, meaning that it is challenging to interpret the reasoning behind their decisions [12]. This lack of interpretability can hinder trust and acceptance, especially in critical applications where safety is crucial [13]. These combined factors make it challenging to achieve in-situ ML using NNs.

Tsetlin Machines (TMs) offer a compelling alternative to NNs [14]. Unlike NNs, TMs provide inherent interpretability, since their operation is based on Propositional Logic [15], allowing for explicit representation of patterns in the data. This characteristic is particularly valuable in scenarios where explainability is crucial for application safety [16]. Their design makes them intrinsically more parallelisable than NNs [17], offering more energy-efficient powering schemes and better scalability. However, it's worth noting that Tsetlin Machines, like NNs, still require the conversion of inputs. Analogue signals need to be converted into a Boolean format for processing by TMs [18]. This conversion step introduces an additional energy cost and the potential for noise and loss of precision, which can impact the accuracy and reliability of the model. Therefore, while TMs offer advantages over NNs for in-situ ML, the need for digital conversion remains a practical consideration that must be addressed to leverage their benefits in real-world applications fully.

Having recognized the challenges, I designed an alternative ML architecture, named *Fuzlearn*, that aims to rectify the issues that are plaguing TMs and NNs. My novel approach takes the Tsetlin Machine as a starting point and addresses the necessity of analogue-to-boolean conversion by re-framing this conversion into the learning process itself. As such, Fuzlearn derives its' strengths from TMs, namely parallelisability and

interpretability, and yet it is a better candidate for solving in-situ ML challenges.

1.2 Hardware challenges

Conventional computing hardware, particularly memory storage, is inadequate for the practical realization of Fuzlearn, specifically, and in-situ machine learning applications in general. When considering the big picture, traditional hardware and architecture may not be the best fit because of their limitations. The prevalent von Neumann architecture separates computing into two parts: the processor and the memory [19]. However, this separation leads to higher latency and energy inefficiency during computing. Essentially, the processor is often idle and waiting for data from the memory, creating a bottleneck known as the von Neumann bottleneck [20]. Unfortunately, this bottleneck cannot be resolved entirely due to fundamental limitations of the architecture [21]. When it comes to the lower/device level, most hardware options are not suitable due to various limitations. Fast memory, like dynamic random-access memory (DRAM), is volatile and can lose stored data due to sudden power fluctuations [22]. On the other hand, non-volatile memory options like solid-state drives (SSDs) and hard disk drives (HDDs) have high energy costs and significant read/write overhead [23]. These issues arise from their reliance on conventional Silicon semiconductor technologies, which lack memristive properties at the material level. Thus, the current options are not suitable for Fuzlearn realisation, which requires exploring alternative approaches and materials for memory storage and usage.

A potential solution has been discovered in Memristors. These innovative electrical components possess a unique ability to modify their resistance based on previously applied bias [24]. Their non-volatile nature makes them safer than RAM for retaining memory. Compared to SSD/HDD, the read-write operations require less overhead. In the bigger picture, memristors can enable a more efficient way of computing, commonly known as in-memory computing [25], which presents itself as an alternative paradigm to von Neumann's architecture tailored to large data tasks, which includes ML. Thus, memristors are a perfect candidate for realizing Fuzlearn hardware.

Among the various possible materials for memristor realisation, Halide Perovskites have been selected as a model material [26]. Halide Perovskites belong to a family of materials characterized by the ABX3 structure [27] and have already demonstrated exceptional performance in photovoltaic applications [28]. In addition to their photovoltaic properties, most Halide Perovskites exhibit inherent hysteresis, which makes them ideal candidates for memristor fabrication [29]. Furthermore, these materials can be processed using low-temperature solution-based techniques [30], which alleviate their fabrication complexity and costs. Despite some research groups utilizing these memristors in specific applications [31], little progress has been made in disseminating Perovskite memristors to a broader audience of electrical engineers and circuit designers. Therefore, I undertook the task of designing a physics-based model of Perovskite memristors that can be readily integrated into circuit simulations, particularly SPICE simulations, catering to the needs of the broader electrical engineering community.

1.3 Thesis aims

In summary, the aims of my project were to conceive, design, and implement a solution for in-situ ML. To achieve this, I created Fuzlearn - a novel ML architecture tailored to analogue signal classification. To realise Fuzlearn in reality, I utilised Halide Perovskite memristors as a building block for Fuzlearn hardware realisation, for which I created a model capable of predicting and replicating actual device behaviour for the purposes of circuit simulation. To help with my modelling, I fabricated real Perovskite memristors and characterised them to investigate the memristive properties of Halide Perovskites which helped to fine-tune the model. Finally, as a final step towards the project's goal, I designed a hardware circuit for the Fuzlearn machine and tested its implementation in circuit simulation software.

1.4 Publications arising

Here is the list of publications made during this project that present key milestones of my PhD:

- "SPICE Modeling and Characterization of Filament Formation Perovskite Memristors" [32]
- "Fuzlearn: A Fuzzy Clusterization based Machine Learning using Learning Automata" [33]

Chapter 2

Background

To achieve in-situ machine learning, addressing the algorithmic problem and the hardware problem is crucial. This chapter delves into these two challenges and provides background information on each. The first section discusses machine learning algorithms that could be potential candidates for in-situ machine learning but have limitations preventing their suitability. The subsequent section explores hardware limitations associated with using the conventional silicon technology on its own and introduces a promising addition called Perovskite memristors.

2.1 Existing ML Algorithms and Their Drawbacks

In in-situ machine learning, selecting the appropriate algorithm and understanding how to execute it are crucial. This section discusses various machine learning algorithms, including classical models, such as decision trees, nearest neighbours, etc. Neural networks (NNs) are the most advanced machine learning approach at the moment, and thus, extra attention is given to their description. Additionally, a novel approach called the Tsetlin Machine (TM) is also presented and discussed. This section highlights the challenges and limitations each algorithm faces for in-situ applications.

2.1.1 Classical ML algorithms and models

In this subsection, I review the already existing and well-established machine learning algorithms and models. In particular, I will present the more classical ML models: their inner workings and their drawbacks that diminish their potential for usage in situ ML.

There are several popular classification algorithms used in machine learning. One of these is the decision tree learning algorithm [34], which is a recursive process of subdividing the training dataset along one of the attributes or inputs that results in the best split of the classes. This iterative splitting leads to a mathematical structure, a tree of 'decisions', which consists of a set of conditional instructions that allow the classification of a new point outside of the original training dataset. However, decision trees are prone to overfitting [35]. In other words, they generate decisions that closely represent the training data rather than the underlying general classification rule.

Another popular algorithm is logistic regression. Here, the training dataset, consisting of datapoints with their assigned binary classes, is fitted into a logistic curve, and the result of training is a function that gives a probability of a new datapoint being one of the two classes [36]. While conceptually simple, the fitting process includes the usage of advanced arithmetic functions, such as exponents, logarithms, etc., both of which require computational capabilities that might not be compatible with in-situ settings.

The k-nearest neighbours (k-NN) algorithm is another classification algorithm that classifies new datapoints by comparing 'distances' between the new point and old training datapoints within the abstract attribute/input space [37]. The class of the new point is assigned in accordance with the classes of the old points closest to it, hence the 'neighbours' in the name. However, every time a classification is performed, each training point must be queried, making its usage more and more costly with more training.

The support vector machine (or SVM) algorithm is another popular classification algorithm, which considers datapoints to be literal points in an abstract attribute/input space [38]. However, unlike k-NN, the model is resolved by finding a surface, usually a plane, that separates the points into two clusters, where each point in a cluster has the

same class. While it doesn't have the same problem of high usage cost as k-NN, it has its own issues, such as being a black box [39, 40]. In other words, it is hard to extract meaning from SVM's results - it cannot explain its own decisions.

To sum up, although each of the common ML models and algorithms has its own strengths, they also have weaknesses and issues that make their use for in-situ ML questionable. The Neural Networks model, which is one of the most commonly used, will be considered and discussed separately in the following subsection.

2.1.2 Neural networks

Neural networks, also known as artificial neural networks, are one of the most common approaches in the field of machine learning and artificial intelligence. Their fundamental design aims to replicate the structure and capabilities of the human brain, rendering them strong and proficient in tackling tasks that were once deemed solely human work [9]. As of this moment, they are the cornerstone of modern ML and AI [41–44]. In this subsection, I will briefly expand upon their working principles and discuss in more detail how the said principles inhibit their utilisation for in-situ machine learning.

Neural networks are composed of interconnected units called neurons or nodes, as can be seen in Figure 2.1. These artificial neurons are inspired by the biological neurons in the human brain [9]. Each neuron receives input signals, performs computations on them, and produces an output signal. Neural networks consist of multiple layers, including an input layer, one or more hidden layers, and an output layer [9]. The input layer receives the initial data, which then propagates through the network layer by layer. Each layer's output serves as the input to the next layer, enabling information to flow through the network. The input layer is responsible for receiving external data and presenting it to the network. The output layer provides the final result or prediction generated by the neural network. The number of neurons in the input and output layers depends on the nature of the problem being solved [45]. The connections between neurons in different layers are represented by weights. These weights determine the strength of the connection and play a crucial role in the network's ability to learn and make accurate predictions [9]. The

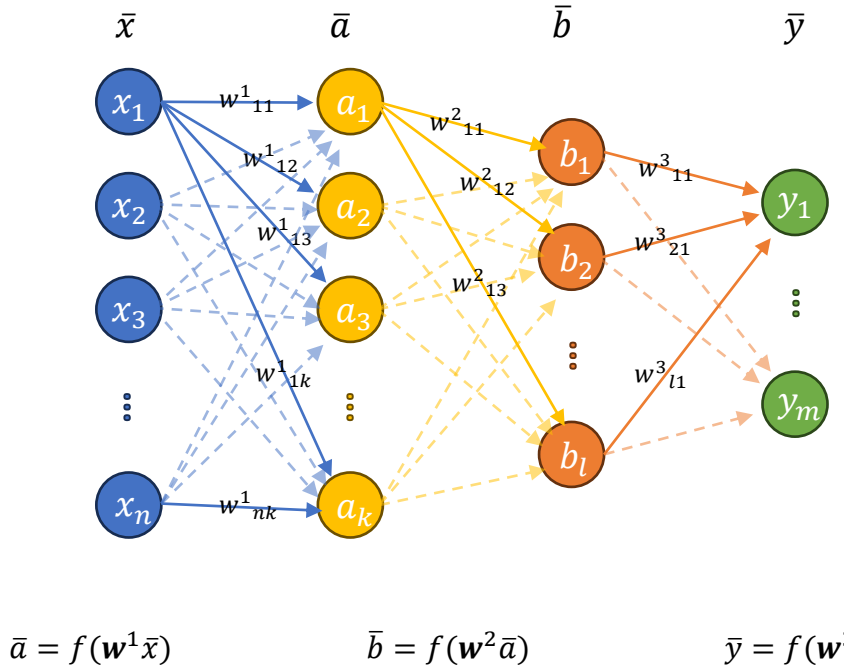


Figure 2.1: Schematic of a neural network. Each circle/node is a neuron. The left-most layer is the input layer, with each input (x_i) a real value. The right-most is the output, $\bar{y}$. Data propagates as the matrix multiplication of the layer values with the corresponding weight matrix ($\mathbf{w}^i$) representing neuron connection weights. This is followed by normalization/activation function, f, to ensure a limited range of possible values. Here, function application, $f(\bar{v})$, is element-wise.

data propagation process involves matrix multiplication between the inputs and weights, followed by normalization or activation functions to limit the range of possible values a neuron can hold and, potentially, introduce non-linearity to the network [9, 45].

Neural networks of larger sizes can learn intricate non-linear connections between input and output data. This increased competence enables them to detect significant patterns from data, depicting highly complex features that smaller networks fail to identify [46]. Nevertheless, the larger size of these networks can lead to heightened computational complexity [46]. This necessitates significant computing resources to train and run large neural networks [47]. Graphics processing units (GPUs) and tensor processing units (TPUs) are often used to accelerate the computations involved in neural network training and inference [48]. These hardware accelerators are designed to manage the intensive matrix operations and parallel computations inherent in neural network processing, providing the

necessary computational power to train and execute large-scale models. The crucial point is that while specialized hardware can offer significant benefits, it's important also to consider the energy consumption that comes with using these powerful resources. GPUs and TPUs are like heavy machinery that require a lot of energy to function [49]. This is especially important to keep in mind in situations where resources are limited or unreliable, such as in remote or in-situ settings.

In addition, neural networks can be quite challenging to understand for humans, as they often exhibit a "black box" behaviour [12, 13]. Despite being able to produce accurate results, the complex interactions between numerous neurons and their weighted connections within the network make it difficult to comprehend their internal workings. Due to this lack of interpretability, it becomes hard to explain why a particular prediction or decision was made, obscuring the network's decision-making process. When neural networks are in charge of critical decision-making, such as those that could pose potential harm to people, their interpretability becomes extremely important [12]. The opaque nature of neural networks raises valid safety concerns in these situations. Such concerns have already been raised with regard to various fields where NNs were applied, such as social [50–52], medical [53, 54], legal [55], security [56, 57], etc. In short, without a thorough understanding of the network's behaviour or appropriate safeguards, unintended consequences or errors may occur, which could lead to undesirable situations.

In summary, neural networks are a powerful tool for automated decision-making. However, they are disadvantaged by the black box problem and large energy requirements, both of which are inherent to NNs' intrinsic nature and, thus, cannot be untangled. Those issues do not disqualify NNs from being applied in situ, per se. But, it is reasonable to seek alternative candidates for in-situ ML, that are more suitable and allow for easier and less requiring development workflow.

2.1.3 Tsetlin Machine - Novel alternative to NNs

The Tsetlin Machine is an intriguing and innovative approach to machine learning, centred around an ensemble of learning automata known as Tsetlin automata [58], specifically

designed to tackle Boolean-to-Boolean classification problems. This unique approach sets it apart from traditional neural networks and other machine learning models, which were discussed in the previous subsections, as it learns through the derivation of a propositional logic expression [15] from observed data points.

The core principle underlying the operation of the Tsetlin Machine is the observation of Boolean data points and the assignment of weights to each combination of input and output based on their frequency of appearance in the training data set [14]. The weight is stored in the machine in the form of the Tsetlin Automata states, which are varied by the means of penalty and reward, see Figure 2.2. This attribute makes it highly suitable for tasks that involve binary decisions, making it particularly efficient for Boolean classification tasks. The TM's overarching working principle is illustrated in Figure 2.3.

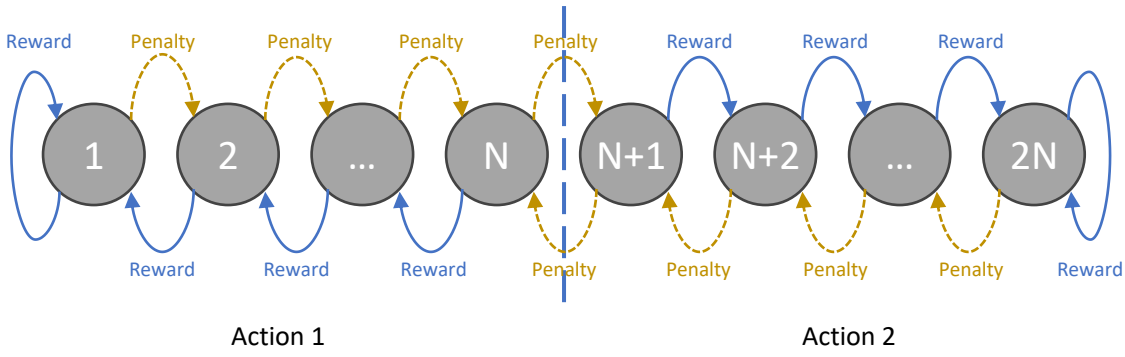


Figure 2.2: Tsetlin automaton schematic - abstract finite state machine capable of performing one of the two actions at each time step. The internal state of the automaton dictates which action will be performed by the automaton: Action 1 for state $\leq N$, or Action 2 when state $> N$. The state of the automaton is controlled by reward/penalty - blue and yellow arrows. Reward reinforces the occurrence of the current action, while penalty forces the automaton to switch the action it performs to the other one.

In it, the states of the automata dictate the final form of the propositional expression. As such, Action 1 and Action 2, as depicted in Figure 2.2, are *exclude* and *include* an input from/into the final propositional expression respectively. A major advantage of the Tsetlin Machine lies in its ability to parallelise computation, which is facilitated by the nature of the final classification procedure. The final output is decided by the majority voting of multiple Clauses, each of which represents a conjunctive product (Boolean AND

operation) of the included inputs, or their respective logical complements. Such design

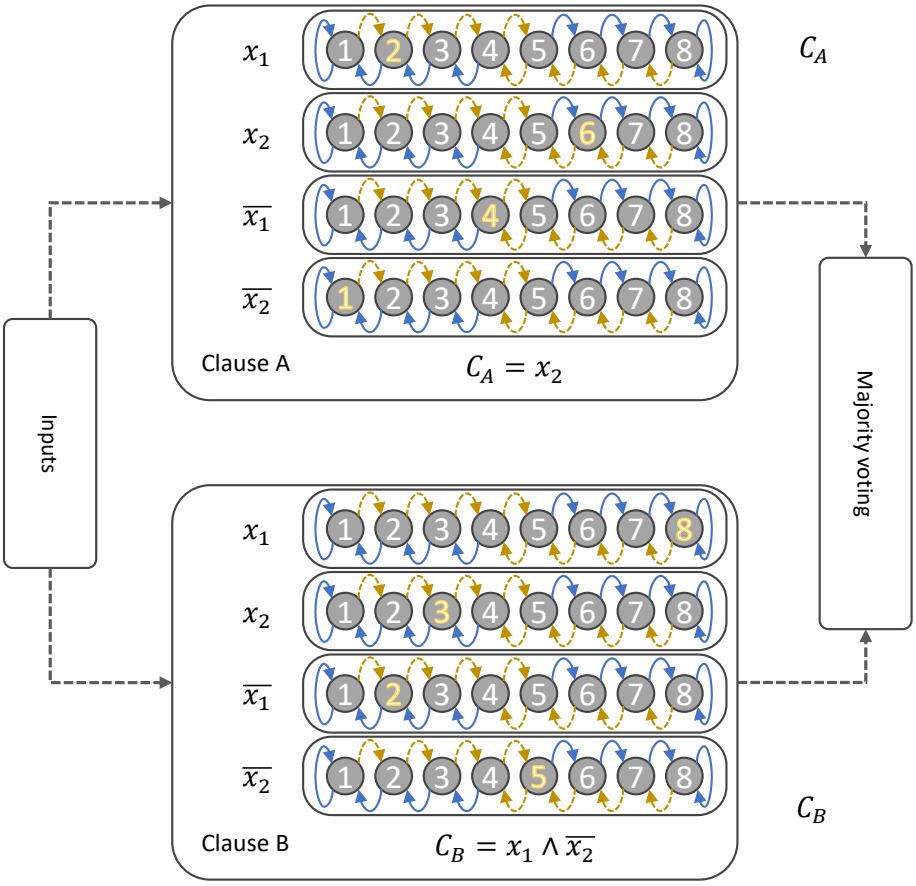


Figure 2.3: Tsetlin machine with two conjunctive clauses with 4 automata each. Two for two Boolean inputs, x_1 and x_2, the other two for their logical complements, $\bar{x_1}$ and $\bar{x_2}$. Each automaton has eight states, $N = 4$. The action the automata perform is *exclude*/*include* the input from/into the propositional expression for the output class. The highlighted state numbers are the current states of each automaton.

allows the learning process to require significantly less computation and, consequently, less energy compared to many other machine learning algorithms [59]. The fact that the Clauses are independent of each other in their decision-making, and as such could be easily parallelised [17], contributes to its efficiency [60], making it valuable in resource-constrained environments and when dealing with large datasets, where computational resources are limited.

Another significant advantage of the Tsetlin Machine is the transparency of its decision-making process. Unlike black-box models like neural networks, the Tsetlin Machine pro-

duces outcomes that are more evident and predictable. Each Clause in the machine represents a conjunctive product of the included inputs. As a result, the model's predictions or outputs are always represented as pure logical expressions of its inputs, which are easily understandable by humans. This level of transparency enhances the model's interpretability and contributes to its safety in critical applications, e.g. in cases of network security [16] and medical applications [61]. Users can comprehend the reasoning behind the decisions made by the model, and as such they can foresee possible problems arising from a TM's decision, which is not the case with NNs.

Nevertheless, the Tsetlin Machine does have certain limitations that warrant consideration. When dealing with real-valued inputs, the data must undergo digitisation or Booleanisation to convert continuous values into discrete Boolean representations. This process demands additional computing power and, in scenarios such as in-situ settings, may require more hardware and energy resources [18]. Moreover, the need for conversion of real-valued data into Boolean form raises concerns about the fundamental principles of machine learning. Conversion, in a way, performs part of the learning process upfront by including the optimisation of the digitisation scheme [62, 63]. I note that this aspect contradicts the essence of machine learning, where algorithms ideally learn meaningful representations directly from raw input.

In summary, the Tsetlin Machine emerges as a superior option for in-situ machine learning compared to neural networks, addressing the issues of computational parallelisation and interpretability that plague NNs. However, the Tsetlin Machine's inherent Boolean nature poses a challenge when dealing with real-valued data in distributed systems like IoT and edge computing. Despite this limitation, it serves as a promising foundation for developing a customized approach to in-situ machine learning, tailored to the specific requirements of these systems.

2.1.4 Summary

Neural networks have emerged as a ubiquitous and powerful tool in various fields, benefiting from extensive research and development. They have demonstrated remarkable

success in tasks such as image recognition, natural language processing, and complex decision-making processes. However, one major drawback of neural networks is their dependence on significant computational resources. Training and running neural networks require substantial processing power and memory, making them computationally expensive and limiting their application in more constrained scenarios.

Tsetlin Machines offer an alternative to neural networks for complex problems, requiring less computational power. Inspired by logical formalism, they efficiently perform pattern recognition using Tsetlin Automata, with the added benefit of transparency and interpretability, overcoming the "black box" issue of neural networks. Nonetheless, Tsetlin Machines necessitate data Booleanisation, introducing complexity and pre-processing in the learning pipeline.

Both approaches have their limitations when applied to in-situ ML. Thus, there is a need for a novel solution that adapts better to limited resources, requires minimal pre-processing, and ensures transparent decision-making.

2.2 Perovskite Memristor for ML hardware

In the pursuit of realizing in-situ machine learning, several challenges arise from the limitations of existing hardware. This section will delve into some of the primary constraints and propose potential solutions to overcome them.

One of the primary obstacles in achieving in-situ ML is the scarcity of suitable hardware. Traditional computing hardware, while capable, is not optimized for real-time, on-device machine learning tasks in remote settings. In-situ machine learning requires hardware that can perform machine learning tasks in real-time, without relying on a server. This hardware must also be energy-efficient, allowing it to operate on low-power sources, and persistent, capable of withstanding volatile scenarios such as power fluctuations without losing any training results.

A major hardware limitation hindering in-situ ML lies in the memory subsystems [10]. The memory is used to store the training/testing data, but also, more importantly, it

is used to store the state/essence of the ML model being used. As such, to realise an energy-efficient and robust in-situ ML platform, one must solve the memory problems.

The most important problem is a lack of compromise between energy efficiency and memory persistence in conventional memory technologies. Conventional hard drive discs and solid-state discs have proven to be energy-intensive regarding read and write operations [23]. This high energy consumption makes them ill-suited for use as main memory, which is crucial for in-situ ML tasks that require frequent and immediate access to data. On the other hand, Random Access Memory (RAM) offers fast data access, making it suitable for in-situ ML applications. However, it is inherently volatile, meaning that data is lost when power is disconnected [22]. For in-situ ML devices that rely on continuous operation in dynamic environments, this volatility poses a serious risk of losing valuable ML progress and data in case of power fluctuations or outages [64].

The ideal memory system would combine the lightweight, fast-access characteristics of RAM with the stability and non-volatility of HDD/SSD, enabling seamless and robust ML processing in real-world scenarios. One promising solution to address this hardware limitation for in-situ ML is the incorporation of memristors [65]. Memristors are a type of next-generation non-volatile memory technology that can retain data even when power is disconnected. This quality makes them similar to storage devices like HDD/SSD but with much faster access times, more akin to RAM. Researchers and engineers can bridge the gap between lightweight, fast-access RAM and stable, non-volatile HDD/SSD by utilising memristors as a memory storage solution in in-situ ML devices. The implementation of memristors has the potential to revolutionize in-situ ML, facilitating real-time processing, reduced energy consumption, and increased reliability, thereby opening up new horizons for deploying ML algorithms in dynamic environments.

As such, it is worth noting that some work has already been done on incorporating memristors into low-power ML hardware architectures. For example, multiPULPly is a low-power memristor-based hardware architecture aimed at hosting and executing Neural Networks [66], which is based on the PULP platform [67].

2.2.1 Memristor - novel electrical component

A memristor, as a novel type of electrical circuit component, possesses intriguing characteristics that set it apart from traditional circuit elements. In strictly technical terms, a memristor establishes a connection between the electric charge passing through it, Q, and the magnetic flux linkage, ψ:

$$M = \frac{d\psi}{dQ}. \tag{2.1}$$

In this equation, quantity M is the memristance, and it has the same units as resistance, $[M] = \mathrm{Wb} \cdot \mathrm{C}^{-1} = \mathrm{V} \cdot \mathrm{A}^{-1} = \Omega$ [68]. Unlike regular resistors, which maintain a constant resistance irrespective of the signal history, memristors exhibit a dynamic resistance that is influenced by the previous electrical stimuli they have encountered [69]. In a more general sense, a memristor can be described as a resistor whose resistance depends on the history of the previously applied bias [24]. This historical dependency of resistance enables memristors to function as memory elements, capable of storing information in the form of resistance values [70]. A typical shape of a memristive device's current-voltage curve would look like Figure 2.4 [71].

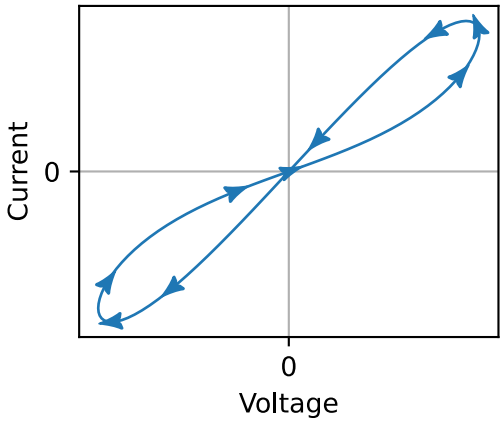


Figure 2.4: A typical shape of a current-voltage curve of a memristive device. Applied bias direction: 0V → Positive V → 0V → Negative V → 0V. The device's resistance follows high → low → high.

Due to these memory-like properties, memristors have garnered significant attention in the realm of memory storage technologies. Specifically, they have been actively explored and employed in the development of Resistive RAM (ReRAM), a memory tech-

nology that utilizes the memristive properties of these components. ReRAM stores data by manipulating the resistance levels of memristors, providing a compelling alternative to traditional memory storage systems [72]. The integration of memristors in memory storage technologies has also found practical applications in the domain of Neuromorphic computing. Neuromorphic computing seeks to emulate the structure and functionality of artificial neural networks by directly mapping theoretical building blocks, such as neurons, into hardware components [73]. In this context, memristors play a pivotal role as they can accurately represent the synaptic weights that connect neurons in the neural network [74, 75]. By emulating the behaviour of biological neurons' synapses, memristors enable Neuromorphic computing systems to accelerate neural network processing and enhance overall performance [76].

Despite the promising advancements in Neuromorphic computing and neural network hardware, the full potential of memristors in in-situ machine learning remains largely unexplored. In-situ ML demands memory technologies that can offer high-speed, low-power, and non-volatile data storage capabilities. Memristors present an intriguing prospect for addressing the memory limitations highlighted in the earlier section and supporting real-time machine learning in constrained environments.

Among the various materials that can be used to create memristors, Halide Perovskites have emerged as one of the most promising options [26]. Halide Perovskite memristors exhibit unique advantages, including excellent scalability, cost-effectiveness in fabrication, and impressive electrical performance. These qualities make them highly attractive for potential in-situ machine learning applications, where lightweight and efficient memory technologies are critical.

As research and development in the field of memristors continue to progress, their integration into in-situ ML systems could bring about transformative advancements in the efficiency and capabilities of real-time machine learning on edge devices and in dynamic settings. By harnessing the memory-like properties of memristors, future in-situ ML applications may achieve unprecedented levels of performance and adaptability, opening up new opportunities for AI-driven solutions in various industries and domains.

2.2.2 Halide Perovskite memristors

Halide Perovskites stand as a highly promising candidate for memristor realisation, drawing on the unique properties of these materials. Belonging to the family of materials with common ionic crystal structure ABX_3 [27], with X representing one of the halogen ions, Halide Perovskites have already exhibited tremendous success in photovoltaic (PV) technologies due to their exceptional light-absorbing capabilities, rendering them an ideal choice for solar cells [28]. Perovskite structure can be seen in Figure 2.5.

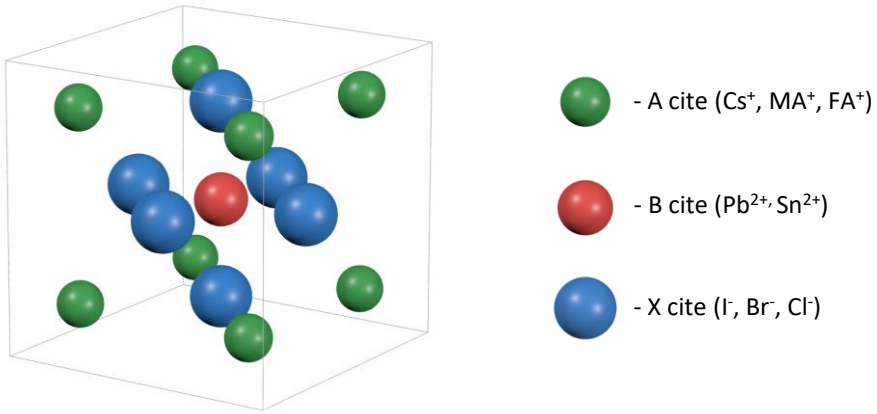


Figure 2.5: Perovskite structure unit cell. A heavy core at B cite, usually occupied by Lead (Pb), is encased in a halide octahedron - X cites - which itself is placed inside of a cube with cations as vertices - A cites.

A standout characteristic of Perovskite materials is their distinctive tendency for hysteresis, a phenomenon frequently encountered while working with Perovskite solar cells [29]. Hysteresis denotes the dependence of a system's output not only on its immediate input but also on its history. Within Perovskite solar cells, hysteresis arises due to the migration of ions and ion vacancies within the bulk of the material when subjected to an electric bias [77, 78]. The presence of ions and ionic vacancies, arising from imperfections in the crystallization process, leads to their movement within the Perovskite material under the influence of an electrical field. Consequently, these dynamic movements cause alterations in the material's electrical conductivity and resistance, giving rise to the observed hysteresis effect in the solar cell's performance. While hysteresis is generally considered undesirable in the context of solar cells as it can affect efficiency and stability,

it proves to be an advantageous property for memristors.

In the context of memristors, hysteresis is embraced and harnessed as a desirable property, often referred to as "memristance." This hysteresis behaviour in Perovskite memristors empowers the creation of memory elements with the unique ability to store data in the form of resistance values [26]. This characteristic makes Perovskite memristors highly attractive and practical candidates for in-situ machine learning applications, where lightweight, efficient and non-volatile memory storage solutions are of paramount importance.

Moreover, the choice of Halide Perovskites as the material for these memristors brings additional advantages to the table. One of the key benefits is their compatibility with low-temperature solution-processing techniques [79], simplifying the fabrication process and making them more amenable to work with compared to other potential memristive materials like Titania (TiO_2), which requires high-temperature annealing [80]. This solution-processing capability also contributes to cost-effectiveness [81], which can potentially enable large-scale production of memristors, thus presenting exciting possibilities for cost-effective, energy-efficient, and high-performance memory technologies in the realm of in-situ ML.

Filament formation

There are two different mechanisms that enable Perovskite memristivity: ionic charge accumulation [82, 83] and filament formation [84, 85]. In the search for advanced memory devices, filament formation - a specific subtype of Perovskite memristance - has become an area of active investigation. Filament formation occurs when a chemically active metallic material, such as Silver (Ag) - the most commonly used active metal - is used as an electrode [26]. These memristors have a more distinct difference between their low-resistive and high-resistive states, with the magnitude of the ratio reaching 10^4 - 10^6 [85]. Additionally, Silver-based Perovskite memristors have superior retention properties, with the memristors staying in a set state for an extended period of time as compared to other metals, such as Gold [84].

Within Silver-based Perovskite memristors, the underlying mechanism driving the memristance behaviour involves the formation of a conductive filament that threads its way through the bulk of the Perovskite material, effectively reducing the overall resistance of the device. The formation of the conductive filament is hypothesised to be intricately connected with the metal-to-Perovskite interface chemistry, wherein Silver atoms can migrate into the Perovskite material [84, 86]. However, the exact nature of the filament is still elusive. Some suggest that it is in fact Silver that creates a continuous filament through the entire device [85, 87], while others claim that it is Iodine vacancies, V_{I^-} that form the filament [84].

Although there is currently no direct evidence about the nature of the filament, the exceptional resistive switching capabilities seen in Silver-based memristor devices indicate its importance. The ability to attain on/off ratios of such magnitude makes Perovskite memristors a highly promising memory technology, with particular potential applications for in-situ ML.

Chapter 3

Fuzlearn - a novel model for machine learning

3.1 Introduction

Addressing the in-situ ML problem requires a customised solution that can connect analogue data acquired from the real world with decision-making based on said data. This chapter introduces my proposed solution, which I call Fuzlearn.

Fuzlearn is a machine learning model specifically designed to excel in in-situ ML settings. Building upon the foundation of the Tsetlin Machine, Fuzlearn introduces several modifications to address the limitations associated with TMs. One notable difference is Fuzlearn's ability to handle analogue inputs seamlessly. Unlike TMs, Fuzlearn eliminates the requirement for an analogue-to-Boolean conversion scheme by incorporating it directly into the learning process. Fuzlearn transforms the input data, identifying suitable boundaries for Boolean classification, and subsequently converting them into a logical expression.

Subsequently, the following sections of this chapter delve into an in-depth exploration of the design principles underlying Fuzlearn. A comprehensive analysis is provided, highlighting the intricacies of its architecture and functionality. Additionally, the chapter proceeds by presenting the software implementation of Fuzlearn, followed by trial testing against a range of real-world analogue-to-Boolean classification problems. Through

these evaluations, the effectiveness and practicality of Fuzlearn's implementation are thoroughly assessed, providing valuable insights into its performance, potential applications, and drawbacks.

3.2 Fuzlearn theoretical framework

This section provides a comprehensive description of Fuzlearn using straightforward mathematical terminology. Initially, the problem that Fuzlearn aims to address is defined, which aids readers in adopting the appropriate mindset. Subsequently, a thorough depiction of each component comprising Fuzlearn is provided, following the ascending order of abstraction.

3.2.1 Problem definition

Before delving into the description of Fuzlearn, it is beneficial to understand what kind of problems Fuzlearn aims to tackle. As such, this subsection is here to assist the reader in embracing a specific mindset and familiarizing themselves with the fundamental concepts necessary for Fuzlearn's definition.

For the purposes of this work, a real-world problem is an analogue-to-Boolean classification problem. That is, an ML model receives analogue inputs, e.g. voltages from sensors or detectors, and is expected to assign a corresponding Boolean class, `True` or `False`, to the said combination of the inputs based on prior training. The assigned class can be later utilised for various purposes, from performing data analysis to controlling circuitry. A practical example of such a problem is a smart, environmentally conscious air-conditioning system. In it, given a set of analogue voltage readings from thermometers inside and outside the building, the task is to determine if a window should be opened (`True`) or remain closed (`False`).

In its design, Fuzlearn considers a particular formal representation of such problems. Any set of analogue inputs that Fuzlearn can receive is represented by an N-dimensional vector, denoted as $\vec{x}$, consisting of normalized real values, x_i, falling within the range

of $[0.0, 1.0]$. Upon receiving such an input vector, Fuzlearn should output a Boolean in return, `True` or `False`. In formal notation, the Fuzlearn's task is to create a mapping

$$C : X^N \to B,$$
$$\text{where } X = \{x : 0 \leq x \leq 1, x \in \mathbb{R}\},$$
$$\text{and } B = \{\text{True, False}\}.$$

With this formal representation of a problem in mind, we can say that a problem can be fully described by its *class distribution* - a function which takes the input vector and returns a Boolean class which a model should return, i.e. a desired class. If we consider the idea of a class distribution from a geometric point of view, the set of all possible input vectors creates a vector space - an N-dimensional cube. As such a class distribution is, therefore, a Boolean field on this vector space, which is akin to a scalar field but with only two possible values. A graphical illustration of this idea is presented in Figure 3.1.

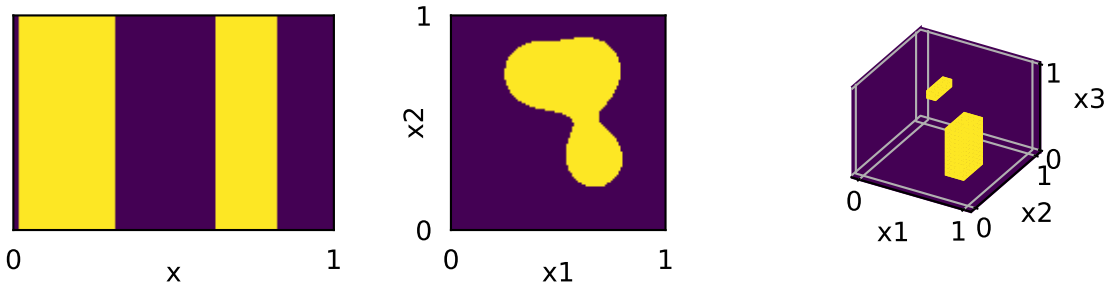


Figure 3.1: Examples of class distributions of real-world analogue-to-Boolean problems. A class distribution is a function that maps all possible input vectors, $\vec{x} \in \mathbb{R}^N$, to their corresponding class, which a machine-learning model should return. Yellow = `True`, Purple = `False`. The examples are class distributions of a 1D, 2D, and 3D problem respectively.

It is noteworthy that since any real value can be mapped bijectively to the $[0.0, 1.0]$ interval. Additionally, any finite number of categorical classes can be encoded either using binary representation (an array of Booleans) or other encoding schemes. Therefore, it is possible to express any machine learning or artificial intelligence problem as a combination of such analogue-to-Boolean classification sub-problems. Consequently, Fuzlearn can be applied to problems of varying dimensionalities by integrating multiple machines into an

ensemble, as will be shown later.

3.2.2 "Fuzzy" Learning automata

Before solving complicated multidimensional analogue-to-Boolean classification problems, it is important to solve the simplest case first. As such, here, I present Fuzlearn Automaton. It aims to solve the simplest classification problem: a single analogue input into a single Boolean class output. Through the lens of Learning Automata theory [88], Fuzlearn Automaton is a type of Learning Automaton with two actions, namely two possible classification outcomes.

Fuzlearn Automaton is the smallest building block of the Fuzlearn model, which is responsible for classifying an individual real value within the interval of $[0.0, 1.0]$ into a Boolean. This classification is accomplished by comparing the input value with two inherent limits: the lower limit, L, and the upper limit, U. The output class of the Automaton, denoted as $A(x)$, is determined based on whether the input value falls within or outside these limits:

$$A(x) = \begin{cases} \textbf{True} & \text{if } L \leq x \leq U, \\ \textbf{False} & \text{otherwise.} \end{cases} \tag{3.1}$$

A graphical representation of the Fuzlearn Automaton's classification function, as well as a diagram symbol, is presented in Figure 3.2. The learning can be facilitated through a training mechanism that dynamically adjusts the limits, L and U, in response to training data.

Unlike other Learning Automata [89] with finite number of discrete states, Fuzlearn Automaton has a continuum of states, each of which is defined by the pair (L, U), where $L \in [0.0, 1.0]$, $U \in [0.0, 1.0]$, and $L \leq U$. As such, the state transitions of Fuzlearn Automaton can be defined as a vector $(\Delta L, \Delta U)$. The actual value of such a vector is subject to the training/learning algorithm, an example of which is presented at the end of Section 3.2.

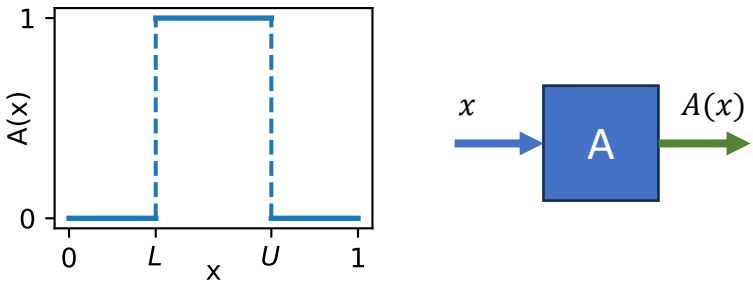


Figure 3.2: Left: Fuzlearn Automaton's classification function. An analogue input, $x \in [0.0, 1.0]$, is classified into a Boolean, `True`/1 or `False`/0 depending on the value of the input. If it falls in between the Automaton limits, L and U, then the result is `True`, otherwise, it's `False`, as described by eq. 3.1. Right: an abstract diagram symbol for Fuzlearn Automaton which is used to describe higher-order abstractions in the Fuzlearn model. The blue-coloured arrow represents an analogue value, Green-coloured arrow represents a Boolean value.

3.2.3 Clause

Most of the real-world problems would require Boolean classification of many inputs, not just one. To classify inputs with multiple dimensions, $N > 1$, it is necessary to employ multiple automata simultaneously. As such, here, the Fuzlearn model borrows the idea of a *clause* from the Tsetlin Machine [14].

A clause is an ordered collection of Fuzlearn Automata. Unlike a single automaton, a clause accepts an N-dimensional array of analogue values as input. Each element (or dimension) of the input array is assigned a dedicated automaton responsible for its classification. The resulting output of the clause is determined by the logical conjunction product of all outputs from each automaton, mathematically expressed as

$$C\left(\vec{x}\right) = \bigwedge_{i=1}^{N} A_i\left(x_i\right). \tag{3.2}$$

A geometric representation of what a 2D clause, i.e. a clause that accepts an input array with two elements (x_1, x_2), looks like is presented in Figure 3.3. In general, for an arbitrary number of input dimensions, N, a clause can be described as an N-dimensional hyper-rectangle, which partitions the input space into points outside (`False`) and inside (`True`) the rectangle. As such, the task of solving a real-world problem is equivalent to the task of overlapping the clause's rectangle over the problem's class distribution.

The boundaries of such a rectangle along each dimension are determined by the limits of the corresponding automaton. Therefore, changing the limits of the automata will change the boundaries of the clause.

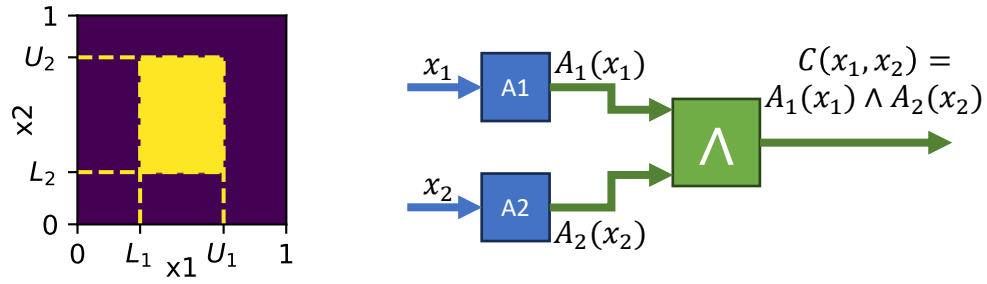


Figure 3.3: Left: A class distribution created by a 2D clause. Each point (x_1, x_2) from $[0.0, 1.0]^2$ square is assigned a Boolean class: Yellow=`True`, Purple=`False`. The resulting `True` rectangle is the intersection of the vertical $(L_1 \leq x_1 \leq U_1)$ and the horizontal $(L_2 \leq x_2 \leq U_2)$ strips. Right: Symbolic diagram of a 2D Fuzlearn clause. $A1$ and $A2$ are individual Fuzlearn Automata assigned to respective inputs, $x1$ and $x2$. The resulting class of the entire clause, $C\,(x_1, x_2)$, is the conjunctive product, i.e. Boolean *AND*, of the Automata outputs.

It is worth noting that clause boundaries are always perpendicular to the input axes. This implies that achieving perfect overlap with the problem's desired class distribution, i.e. achieving 100% accuracy becomes unattainable in cases where the class distribution possesses geometric features that are non-perpendicular to the input axes. This issue will be discussed in more detail in the next sections of this chapter.

3.2.4 Machine

As alluded to in the previous subsections, it is highly unlikely that a real-world problem can be accurately solved with a single rectangular clause. As shown in Figure 3.4, a problem's class distribution can have non-rectangular features and/or the distribution can contain multiple non-joint clusters. In both cases, the overlap between the rectangle and the problem's class distribution is inherently limited. In terms of solving a real analogue-to-Boolean classification problem, this issue can be viewed as an insurmountable limit of the maximum achievable accuracy.

The solution to this issue lies in employing multiple clauses at the same time. Each of the clauses can be responsible for a particular subset of the input space. Such an

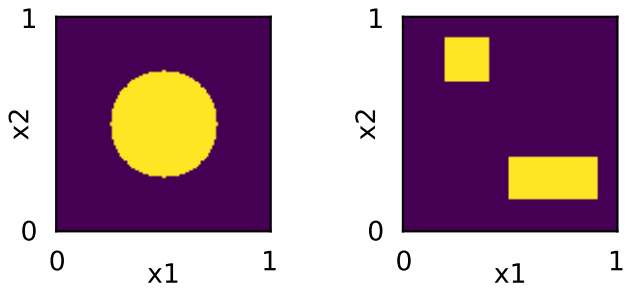


Figure 3.4: Examples of problem class distributions which cannot be overlapped with a single clause/rectangle. Each problem is 2D, i.e. the input array has $N = 2$ elements, (x_1, x_2). The colour of each (x_1, x_2) point represents its desired class. Yellow = `True`, Purple = `False`. Left: A problem can have features non-parallel with input axes. Right: A problem can have multiple discontinues `True` clusters.

ensemble of clauses in Fuzlearn is referred to as the *Machine*. The inspiration for such a design choice comes from the Tsetlin Machine [14]. Evidently, multiple clauses can be used to approximate multi-clustered problems, such as the one depicted in Figure 3.4(right). Additionally, this approach can be utilised to help with the issue of non-perpendicular class distribution features. Figure 3.5 presents a graphical representation of how an ensemble of multiple 2D clauses can be used to tackle a problem with non-rectangular features. In it, a circular class distribution, identical to that from Figure 3.4(left), is tiled with multiple rectangles, achieving coverage better than what could be achieved with a single clause.

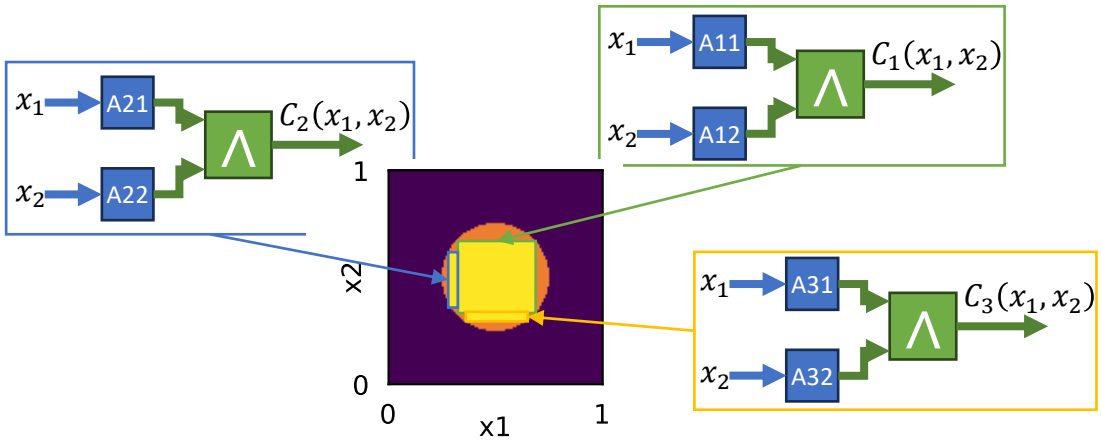


Figure 3.5: Ensemble of clauses deployed to tile a circular 2D classification problem. Input space has $N = 2$ dimensions, i.e. $\vec{x} = (x_1, x_2)$. The colour of each point represents its class. Orange = problem's desired `True`, i.e. *True Positive*, Yellow = machine's `True`, Purple = both problem's and machine's `False`. The machine consists of 3 clauses, each of which is represented with a yellow rectangle. Different outline colours indicate different clauses.

3.2.5 Operating protocol

Up until this point, Fuzlearn, as it was described in the previous subsections, serves as a mere backbone or framework, lacking the capacity for learning. To enable actual learning, it requires a scheme/algorithm which is responsible for how clauses are created and/or updated. The name for such an algorithm or set of algorithms is the *operational protocol*. Such a protocol handles clause allocation and adjustment of the automata limits in response to the provided training data, thereby facilitating the learning process of the model.

In general, an operating protocol is not a concrete part of Fuzlearn. That is, many different schemes and clause-updating procedures can be applied to Fuzlearn to suit end users' needs. Here, I present an example of an operating protocol which implements supervised learning [90].

The important questions that a protocol must answer are "Given a training point, what should Fuzlearn do exactly? Should it instantiate a new clause? Or should it update an already existing one? If that's the case, which one exactly?" To have a concrete answer to these questions, this exemplar protocol draws inspiration from the "Divide and Conquer" approach [91]. Essentially, the entire input space is divided into a grid consisting of K^N cells, where N represents the number of input dimensions and K denotes the number of subdivisions per dimension. Within this grid, only a single clause can be associated with each cell. In other words, a cell can either be occupied by a clause or remain vacant.

The inference process, i.e. the act of using the model to classify the input into a class based on learned knowledge, operates as follows. If the input point falls into an empty cell then the output class `False`. If the input point falls into an occupied cell, but otherwise lies outside of the clause's boundaries within that cell, then the output is also `False`. Only if the input point falls into an occupied cell and it is within the boundaries of the corresponding clause then the output is `True`.

The learning process, presented in Table 3.1 as pseudo-code, can be summarised in the following manner. Initially, the whole grid is empty - no cell contains a clause. Then, the

Table 3.1: Operating protocol: pseudo-code of the learning algorithm. The training process is a game, wherein the Fuzlearn machine receives training points sampled from the problem's desired class distribution and adjusts its clauses' boundaries accordingly. After each training point, the machine tries to change its definition in order to be correct for the current training point. As the training process continues, the machine's class distribution converges to the problem's desired class distribution. Initially, all cells are vacant, i.e. *Clauses*.keys is an empty list.

List of variables

Name	Type	Comment
N	**Integer**	Number of dimensions
K	**Integer**	Number of subdivisions
$\vec{x}$	array of **Real**	Input vector
Class	Boolean	Desired output class
$\vec{c}$	array of **Integer**	Cell coordinates
Clauses	map [$\vec{c}$ →Clause]	Pool of employed clauses

Training algorithm

```
#   Code
 1   for i ∈ [1, N] do
 2     if x_i < 1.0 then
 3       c_i ← ⌊x_i × K⌋
 4     else if x_i = 1.0
 5       c_i ← K − 1
 6     end if
 7   end for
 8   if c⃗ ∈ Clauses.keys then
 9     if Clauses[c⃗].classify(x⃗) = Class then
10       do nothing
11     else
12       if Class = True then
13         Clauses[c⃗] extend boundaries to include x⃗
14       else if Class = False then
15         Clauses[c⃗] retract boundaries to exclude x⃗
16       end if
17     end if
18   else if c⃗ ∉ Clauses.keys then
19     if Class = True then
20       Clauses [c⃗] ← new Clause covering cell c⃗
21     else
22       do nothing
23     end if
24   end if
```

learning proceeds iteratively for each training point. At the beginning of each iteration, Fuzlearn performs the inference. If the inference output is correct, i.e. it matches with the training point desired class, then the machine remains unchanged as there is no error

to correct - no new information to learn. However, if the inference is incorrect then the algorithm updates the machine depending on the circumstances of the misclassification. If a to-bo-classified-as-`True` point falls into an empty cell then a new clause is spawned fully occupying this cell space. If the cell, where the point falls into, is occupied then the algorithm performs one of the clause-updating subroutines.

Table 3.2: Operating protocol: pseudo-code of the clause update subroutines. Clause extension iteratively checks every automaton, and those, that misclassify the training point, are updated with the appropriate boundary placed on the position of the point. Clause retraction chooses one of the automata whose boundary is the closest to the point and moves the boundary to the point's position $\pm\Delta$. Here, Δ is a very small yet distinguishable number, acting as a hyperparameter of the operating protocol.

List of variables

Name	Type	Comment
N	**Integer**	Number of dimensions
Cl	**Clause**	Clause to be updated
Cl[i]	**Automaton**	i-th Automaton of Cl
$\vec{x}$	array of **Real**	Training input vector
Δ	**Real**	A small number $= 2 \times 10^{-7}$
dists	array of **Real**	Array of distances from $\vec{x}$ to all boundaries of Cl
j	**Integer**	Index of the smallest element of $dists$

Clause extension

```
#  Code
1  for i ∈ [1, N] do
2    if x_i < Cl[i].L then
3      Cl[i].L ← x_i
4    else if x_i > Cl[i].U then
5      Cl[i].U ← x_i
6    end if
7  end for
```

Clause retraction

```
#  Code
1  for i ∈ [1, N] do
2    dists[i] ← min(|Cl[i].L − x_i]|, |Cl[i].U − x_i]|)
3  end for
4  j ← index of the smallest element of dists
5  if dists[j] = x_j − Cl[j].L then
6    Cl[j].L ← x_j + Δ
7  else if dists[j] = Cl[j].U − x_j then
8    Cl[j].U ← x_j − Δ
9  end if
```

Table 3.2 presents the pseudo-code of the updating subroutines for the clauses' bound-

aries. In each case, the subroutines are designed such that the misclassification issue is rectified while keeping the overall change to the original class distribution of the clause to a minimum. In case the clause must be extended, i.e. a to-be-classified-as-`True` training point lies outside the clause's boundaries, the algorithm iterates through each automaton, and, for those automata that misclassify their respective component of the input, it updates one of the limits, L and U, to be equal in value to the respective input element, ensuring the point is classified as `True`. A schematic of a two-dimensional clause extension is presented in Figure 3.6.

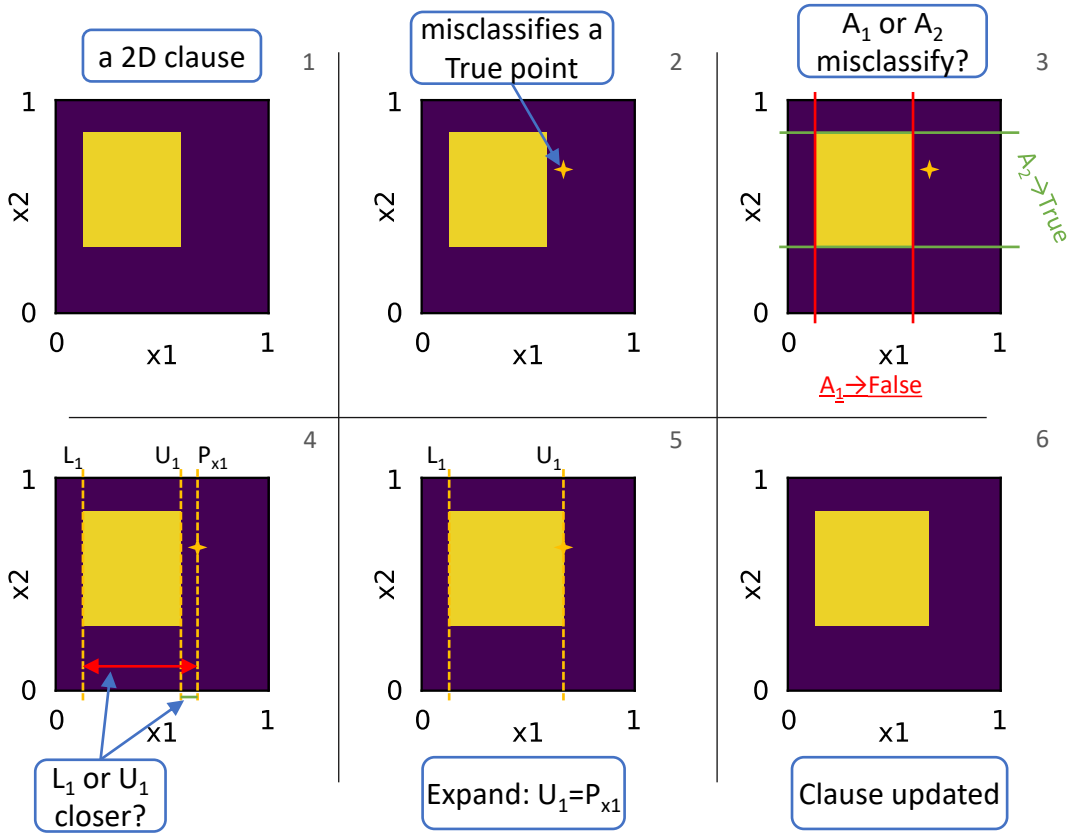


Figure 3.6: Example of a clause expansion update. A 2D clause receives a `True` point and misclassifies it, i.e. the point lies outside of the clause's original boundaries. To fix the misclassification, the update process involves iterating over all automatons, that constitute the clause and that misclassify the point, and extending their limits, L or U, such that the point on the updated boundary. This way the misclassification is fixed with the minimum change to the class distribution of the entire clause.

In the retraction case, where a to-be-classified-as-`False` point falls within the clause's boundaries, the algorithm identifies the automaton/dimension with the nearest boundary

to the point and retracts it such that the point lies just outside. To achieve this 'just outside' boundary situation, the chosen boundary is first moved to the position where the point is located, and then it is moved again in the same direction by a small amount, which is referred to as Δ in the pseudo-code. This quantity can be thought of as a hyperparameter of the operating protocol.

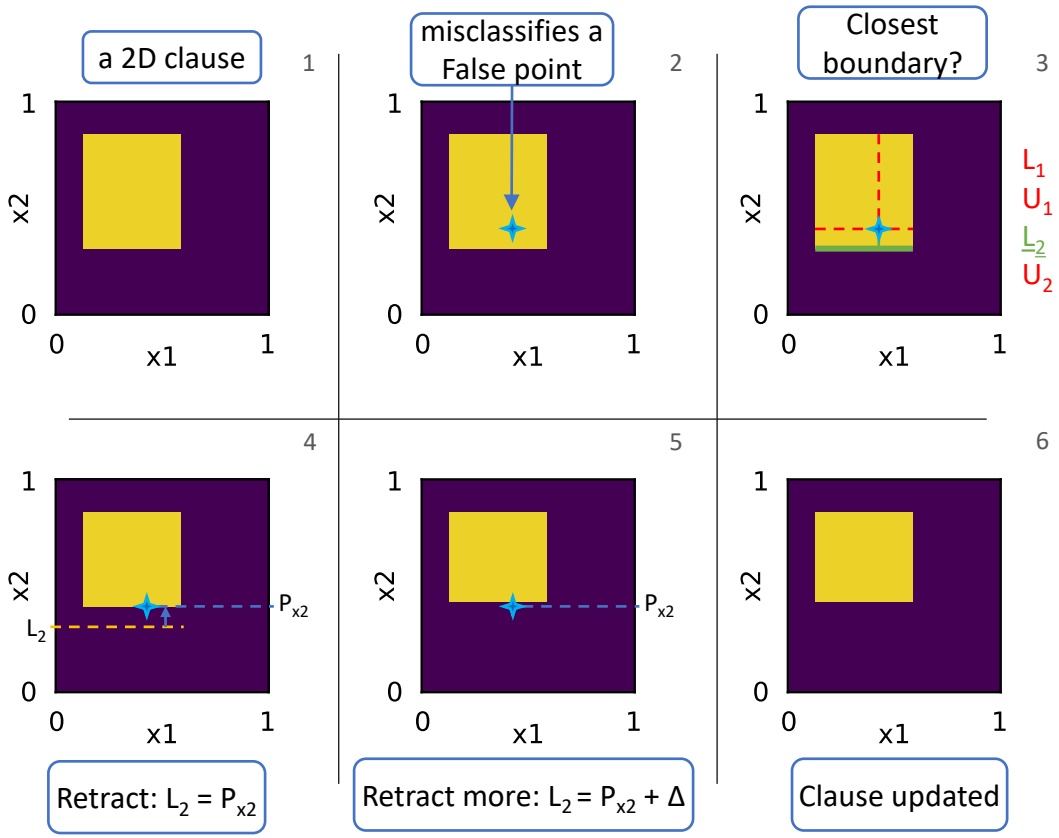


Figure 3.7: Example of a clause retraction update. A 2D clause boundary is retracted after a `False` point was misclassified, i.e. the point was included in the clause's original boundaries. Unlike in the clause expansion update scheme, for the retraction, a change of only one of the automaton's limits is enough to fix the misclassification. As such, the boundary that is the closest to the point is updated to ensure a minimal change to the clause's class distribution. The said boundary is moved to the point and a little bit more.

An important thing to note is that the operating protocol does not differentiate between training points at the beginning of the training procedure and training points at the end. As such, any training that has already been received by the machine could be undone by future training. This presents a couple of issues. First, it means that the final trained state of the model depends on the order in which training points are presented. Second, it

is possible for the useful prior training of the model to be completely undone by a 'rogue' new point. These and other issues will be investigated in greater detail in the following section of this chapter.

However, it's important to reiterate that the operating protocol is not *'set in stone'*. Issues that arise in Fuzlearn's work can be attributed to the specifics of this protocol and not the overall 'spirit' of Fuzlearn. I believe these issues can be amended by a refinement of the protocol; however, I recognise the fact that designing an optimal operating protocol could be quite a challenging endeavour.

3.3 Software implementation and its testing

To prove that Fuzlearn, governed by the presented operating protocol, is indeed capable of machine learning, and to gain insight into its performance and some of its potential drawbacks, I implemented it in software and tested it against real-world problems. I utilized Python for Fuzlearn's implementation since it is a concise programming language which enables quick and easy software development [92].

To test the said implementation, I obtained multiple datasets from an open repository dedicated to machine learning datasets [93]. The datasets were randomly split into training and testing subsets in an 80%:20% ratio, respectively. The training subset was utilized to train the model, while the accuracy of the model was measured against the testing set after each iteration of the training process. To ensure robustness and mitigate the impact of specific random number generator seeds used for dataset splitting, the training was repeated multiple times, providing a more comprehensive understanding of the model's behaviour while avoiding any fortuitous or unfavourable outcomes.

I opted to evaluate Fuzlearn on various problems to assess its validity as an ML model. Initially, I chose to test it on the Iris flower dataset, which is often considered the introductory example in machine learning. Although this dataset is not extensive (consisting of 4 inputs and 150 data points), it serves as a sufficient benchmark to confirm the validity of a machine-learning model.

Subsequently, I proceeded to test Fuzlearn on progressively more challenging problems. The first was the HTRU2 telescope survey dataset, followed by the OZONE ground-level dataset. These datasets were specifically selected because they represent real-world scenarios with analogue inputs requiring classification. Compared to the Iris dataset, these two present higher dimensionalities with 8 and 72 analogue inputs, respectively.

To gain further insights into its capabilities and potential limitations, I conducted tests on Fuzlearn using a suite of artificial problems. I evaluated its performance on the Circle problem, which involves non-perpendicular features, to observe how the model handles such scenarios. Additionally, I focused on cases where noise could impact the results. For this purpose, I conducted tests on three different Noisy XOR problems, each presenting varying types of noise, that might be encountered *'in the wild'*.

3.3.1 Iris flower problem

To prove that Fuzlearn is even capable of performing ML, I first tested it against a rather simple yet sufficient problem, which is the Iris flower problem. The Iris flower problem referred to as the Iris dataset or Fisher's Iris dataset, is a well-known and widely utilized benchmark in the realm of machine learning and pattern recognition [94]. Its primary objective is to categorize iris flowers into three distinct species by analysing their physical characteristics. The dataset comprises measurements of four distinct attributes, namely sepal length, sepal width, petal length, and petal width. These measurements are collected from 150 iris flowers, with 50 samples representing each of the three species: Setosa, Versicolor, and Virginica. The aim of the Iris flower problem is to construct a machine-learning model that can effectively classify iris flowers based on their attribute measurements. This problem is often employed as a fundamental and introductory illustration in the field, frequently used to showcase diverse classification algorithms and techniques.

As explained earlier in the "Problem Definition" subsection 3.2.1, it is feasible to divide a problem with any finite number of classes into a series of problems, each having only two classes. In this approach, instead of utilizing binary encoding, I extract each class as a separate problem. Consequently, there are now three distinct datasets and three

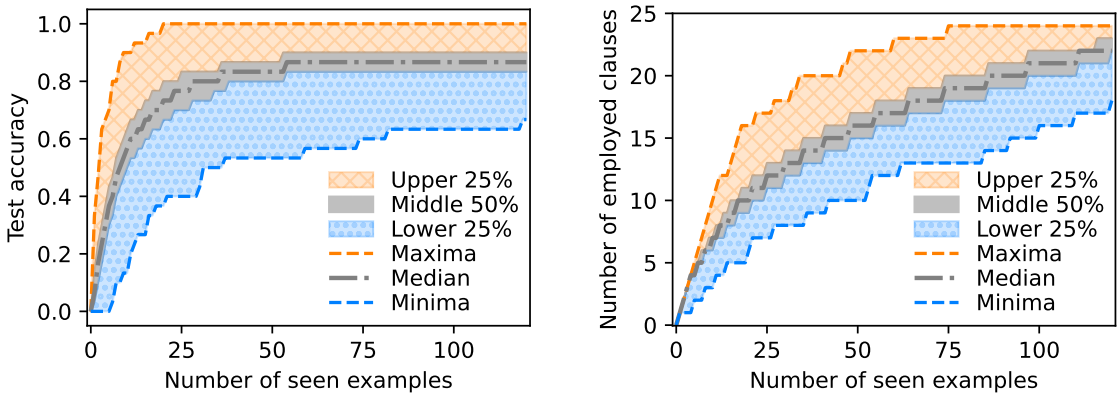


Figure 3.8: Fuzlearn training on the Iris flower dataset. The number of subdivisions per input dimension $K = 3$. A single training cycle consisted of the iterative training on 100 randomly sampled points from the dataset, and tested after each training point on the rest points - testing sample. Training cycles were repeated 1500 times and the results were plotted into quartiles for better statistical representation of Fuzlearn performance. **a)** Test accuracy dynamics. **b)** Growth of the total number of employed clauses.

individual machines assigned to solve them. The new datasets retain the same inputs, but the classes are categorized as either `True` if the class in the original dataset matches the corresponding class of the machine, or `False` if the class differs. Eventually, the overall accuracy is evaluated by considering all three machines together as an ensemble. This means that the ensemble is deemed correct only when all three machines produce correct outputs simultaneously.

Results and Discussion

Figure 3.8(a) illustrates the incremental improvement in test accuracy as the ensemble receives additional training examples. The entire training session was repeated 1500 times, randomizing the split and order of the training and test subsets. The results from all individual sessions were analysed to derive statistical metrics such as the inner/outer 50% range and the median with extremes, which provide an overview of the overall trend. It is evident that the ensemble can achieve an accuracy of 86% after being exposed to the training dataset at least once. However, it is important to note that this does not imply seeing the entire dataset, but rather 4/5 of it. To estimate the resource consumption of the combined machine ensemble, the total number of employed clauses was monitored

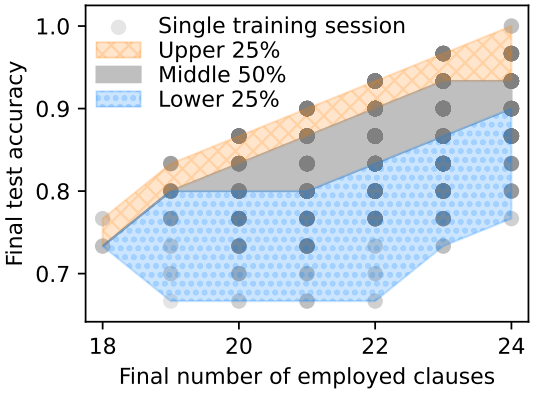


Figure 3.9: Fuzlearn training on the Iris flower dataset. The number of subdivisions per input dimension $K = 3$. The graph represents the correlation between the final achieved test accuracy by Fuzlearn as a function of the final number of employed clauses. Semi-transparent grey points - individual training sessions (1500 in total). Quartiles provide a better representation of the sessions' results distribution.

throughout the training process, as depicted in Figure 3.8(b). The graph clearly demonstrates that even after the initial round, the ensemble can successfully learn to solve the Iris problem using a modest number of clauses, specifically ≤ 24, with an average of ≈ 22.

In general, these results indicate the effectiveness of Fuzlearn in performing machine learning tasks. Another crucial aspect to consider is how the availability of resources impacts the final accuracy. This investigation is essential as Fuzlearn would typically operate under limited resources in real-world, in-situ machine learning scenarios. Therefore, it is crucial to gather at least some preliminary information regarding potential resource management. To address this, Figure 3.9 was generated. It illustrates the relationship between the final test accuracy in a single learning session and the corresponding number of employed clauses. Two significant conclusions can be drawn from this figure. Firstly, an increase in the number of employed clauses, which can be potentially achieved by increasing the number of subdivisions per dimension, can enhance the maximum attainable accuracy. However, upon examining the overall vertical distribution of the data points, the second conclusion becomes evident. Simply increasing the maximum achievable accuracy does not guarantee an increase in the actual accuracy. There are various factors at play, with the composition of the training dataset being a prominent factor. Overall, these findings shed light on the significance of resource allocation in relation to the final

Table 3.3: Accuracy comparison of different ML models trained on the Iris flower dataset. Multiple accuracy values indicate accuracies reported in different sources.

Model	Test accuracy %	Source
Fuzlearn	86.7%	This work
Decision tree	93.9%	[95, 96]
Logistic Regression	91%	[96]
k-NN	92.8%, 93%	[95, 96]
SVM	96%	[96]
Neural Net	93.1%, 95.9%	[95, 97]
TM	95.1%, 96%	[98, 60]

accuracy of Fuzlearn. It emphasizes the need to consider multiple factors beyond the sheer number of clauses, with the training dataset itself playing a pivotal role in determining the model's performance.

To put Fuzlearn's results into context, Table 3.3 presents the accuracy comparison of Fuzlearn with other machine learning models. From this table, it is clear that Fuzlearn underperforms as compared to other ML algorithms. However, it is worth noting that many authors did not specify the number of trials conducted for each model. As such, it is not clear whether their results were 'lucky' or not.

When comparing Fuzlearn with most other models, it is difficult to make valid resource requirement comparisons as it can be like comparing apples and oranges. However, it is possible to compare Fuzlearn with Tsetlin Machine due to their structural similarity. As described in Chapter 2, TM is comprised of clauses each of which represents an ensemble of Tsetlin Automata. To make a Tsetlin Automaton, one would need a device to store a state variable. A memristor can do it. Therefore, it is possible to compare how many memristors would be needed to make Fuzlearn and TM respectively. Note, that this is a 'back of an envelope' calculation and should be taken with a grain of salt. From Figure 3.9, Fuzlearn required a maximum of 24 clauses. Each Fuzlearn clause consists of 4 Fuzlearn Automata, each of which represents one input. Additionally, each Fuzlearn Automata has 2 state variables. Therefore, the total number of memristors required for Fuzlearn is $24 \times 4 \times 2 = 192$. For Tsetlin Machine, as described in [98], the total number of memristors would be 100 clauses $\times$ 4 real-valued inputs $\times$ 5 bits per input $\times$

2 Tsetlin Automata per bit = 4000 memristors, which is ≈ 20 more than what is needed for Fuzlearn. As for TM, proposed in [60], the total number of memristors would be 3 classes $\times$ 20 clauses per class $\times$ 16 binary inputs per clause $\times$ 2 Tsetlin Automata per binary input = 1920 memristors, which is 10 more than Fuzlearn. As such, it can be stated that while slightly compromising the accuracy, Fuzlearn requires at least an order of magnitude fewer components for realising in hardware. Alternatively, it can be stated that if given more resources, Fuzlearn may potentially achieve higher accuracy and be on par with other ML models.

3.3.2 HTRU2 dataset

To test how Fuzlearn would handle a more challenging problem, I trained it on the HTRU2 dataset. Compared to the Iris dataset, the HTRU2 dataset presents a greater challenge as it contains twice as many inputs, with $N = 8$.

The HTRU2 dataset provides a collection of pulsar candidates obtained from the High Time Resolution Universe Survey. Within this dataset, the legitimate pulsar examples form the minority class labelled as True, while the majority class consists of spurious examples labelled as False. Specifically, the dataset consists of 16,259 spurious examples caused by RFI/noise, and 1,639 genuine pulsar examples [99].

Additionally, the imbalanced distribution of True and False classes poses a problem when evaluating performance. In this case, the total number of True datapoints accounts for less than 10% of the dataset. Consequently, a machine that simply classifies all points as False would achieve an accuracy of 90%. To address this issue, an alternative metric called the True Positive Rate (TPR) is adopted to measure overall performance. The TPR represents the accuracy of True points only, reflecting the probability of a point classified as True actually being True.

It is important to note that the testing approach for the HTRU2 dataset varies from that of the Iris dataset. Rather than conducting tests after each individual training example, the testing phase occurs once the entire training dataset has been processed. In the broader machine learning community, this point is commonly referred to as the completion

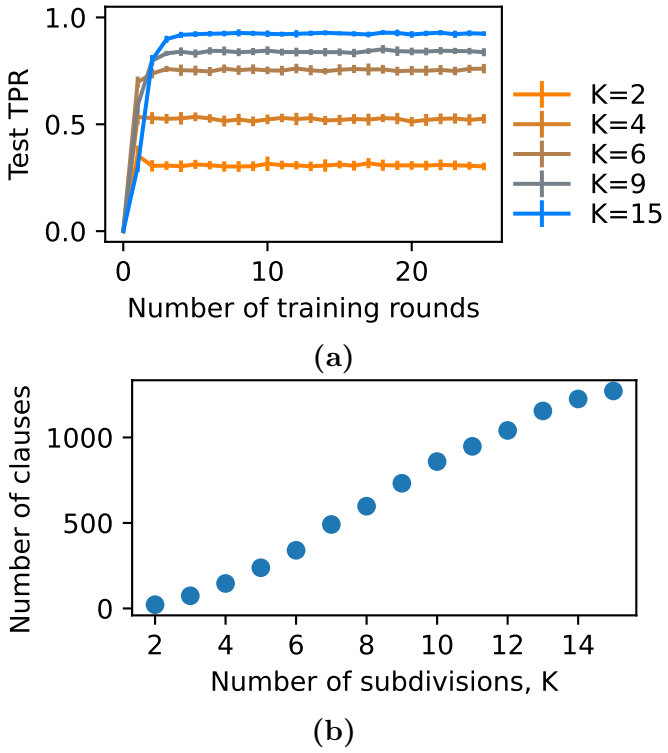


Figure 3.10: Fuzlearn training on the HTRU2 dataset. Graphs present averaged data over 25 individual training sessions. A single session consists of rounds, where in each round Fuzlearn sees the entirety of the training dataset, randomly sampled from the original dataset. **a)** True Positive Rate (i.e. accuracy of `True` points only) dynamics for various numbers of subdivisions, K. Error bars represent the standard deviation of all individual sessions. Maximum achieved TPR rises with K. **b)** Final number of clauses vs number of subdivisions K.

of a "training round" or "epoch." This testing strategy aims to examine the impact of the training dataset on the final accuracy of the machine. To mitigate the limitations imposed by a single training dataset, the machine is given the opportunity to learn from multiple training datasets while retaining previous knowledge. As before, multiple training sessions were performed to gain insights into the general trends. However, due to the dataset's size, only 25 individual training sessions were conducted in this case.

Results and discussion

Figure 3.10(a) illustrates the relationship between the True Positive Rate (TPR) and the number of training rounds for various values of dimensional subdivisions, denoted as K. The quantity K was adjusted to indirectly regulate the number of employed clauses by

modifying the total available cells. The graph demonstrates that Fuzlearn exhibits the ability to achieve high TPR values as the number of subdivisions increases.

Furthermore, Figure 3.10(b) displays the number of clauses employed in each case. The graph showcases a distinct sigmoid curve-like pattern; however, the underlying cause of this behaviour remains unclear. It is possible that this phenomenon is a general characteristic observed across various problems, or it could be specific to this particular problem.

In summary, the testing conducted on this problem demonstrates that Fuzlearn possesses the capability to solve real-world problems effectively, extending beyond simple toy problems like Iris. It serves as a compelling example of an in-situ machine learning application. For instance, a Fuzlearn hardware machine could potentially be deployed on a satellite telescope to facilitate on-board decision-making. This approach would reduce the latency between making initial observations and determining whether to allocate additional time and resources for further investigation of potential regions of interest in space. Moreover, such a system has the potential to reduce energy costs by eliminating the need for expensive data transfers between the satellite and the control centre on Earth.

3.3.3 Ozone level dataset

Now, with HTRU2 being solved, the question is 'What if we go even further? When would Fuzlearn break down?'. Thus, the following dataset was presented to Fuzlearn.

The dataset at hand is the ground ozone level dataset, consisting of two subsets, each with approximately 2500 datapoints [93]. Each datapoint is associated with 73 attributes. Upon merging the two subsets and eliminating any datapoints with missing data, the total number of datapoints reduces to 3695.

The primary challenge of this dataset is to classify each datapoint as either an 'ozone day' (True) or a 'normal day' (False). Similar to the HTRU2 dataset, this dataset suffers from significant class imbalance, with only 185 True points in total. Consequently, the TPR is monitored instead of accuracy to assess performance. Overall, the testing process for this dataset closely resembles that of the HTRU2 dataset, with the exception of the number of individual training sessions. In this case, 100 sessions were conducted due to

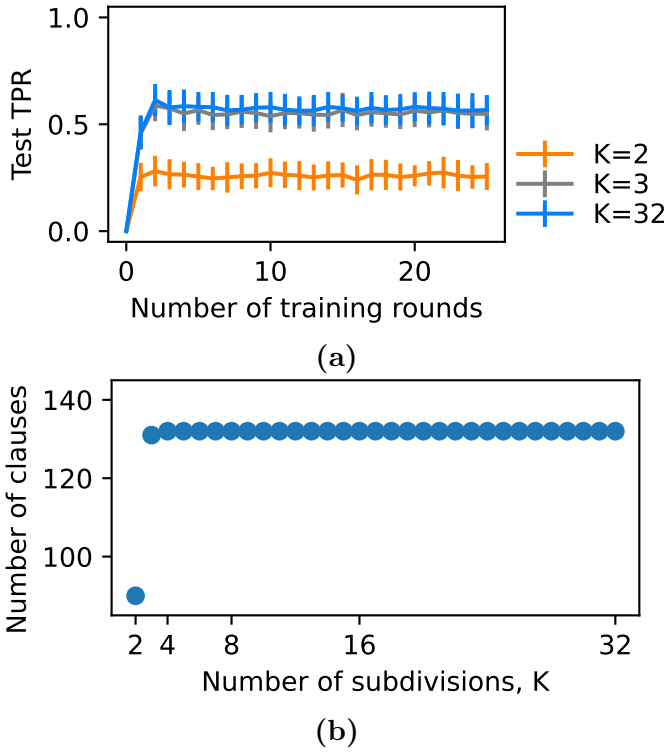


Figure 3.11: Fuzlearn training on the Ozone dataset testing. Graphs present averaged data over 100 individual training sessions. A single session consists of rounds, where in each round Fuzlearn sees the entirety of the training dataset, randomly sampled from the original dataset. **a)** True Positive Rate (i.e. accuracy of `True` points only) dynamics for various numbers of subdivisions, K. Error bars represent the standard deviation of all individual sessions. The maximum achieved TPR is limited at $\approx 50\%$. **b)** Final number of clauses vs number of subdivisions K. The number plateaus after $K = 3$

the relatively smaller number of datapoints.

The rationale behind selecting this particular dataset is straightforward. It represents a real-world problem involving analogue input values that need to be classified into Boolean categories. Furthermore, the dataset poses an additional challenge with its 73 attributes, increasing the complexity of the classification task.

Results and discussion

In Figure 3.11(a), the TPR versus training rounds for the Ozone level dataset is depicted. It is evident that, unlike the HTRU2 case, the machine struggles to achieve a high TPR. Additionally, Figure 3.11(b) illustrates the tracking of the final number of employed clauses. Notably, when considering the number of clauses, it is intriguing to observe that

$185 \times 0.8 = 148$ corresponds to the expected number of *True* datapoints in a single training batch. Remarkably, this value is slightly above the maximum number of employed clauses, suggesting that each datapoint likely occupies its own cell alone.

This circumstance can be attributed to the dataset's high dimensionality. When certain attributes of two or more True points are close together, the remaining attributes can significantly increase the geometric distance in the N-dimensional input space, resulting in substantial separation. As a result, it is highly probable that each individual point resides in its own cell, resulting in a lack of associated clauses for *True* points from the test batch. Thus, this constitutes a limitation in Fuzlearn, as it cannot discern the meaningful attributes from the irrelevant ones that contribute nothing to the classification task. This factor likely plays a pivotal role in limiting the final TPR achieved in this test.

3.3.4 Circle problem - Non-perpendicular features

After proving that Fuzlearn is capable of ML, it is time to get a closer look at the drawbacks that Fuzlearn has or may have. Through this particular problem, my objective is to delve deeper into the issue that arises due to the non-perpendicularity between the features of the desired class distribution and the input axes. Here, due to the problem being 2D, I visualise both desired and converged class distributions graphically for a clearer perspective.

The Circle problem is a toy problem designed to explore potential challenges faced by Fuzlearn. In this specific scenario, the Circle represents a two-dimensional circle in the input space, as depicted in Figure 3.12(a). The input consists of two dimensions, and the classification is determined based on whether the point falls within or outside the circle. The appropriate logical expression defining the class is provided as:

$$C(x, y) = \left((x - 0.5)^2 + (y - 0.5)^2 \leq 0.25^2\right). \tag{3.3}$$

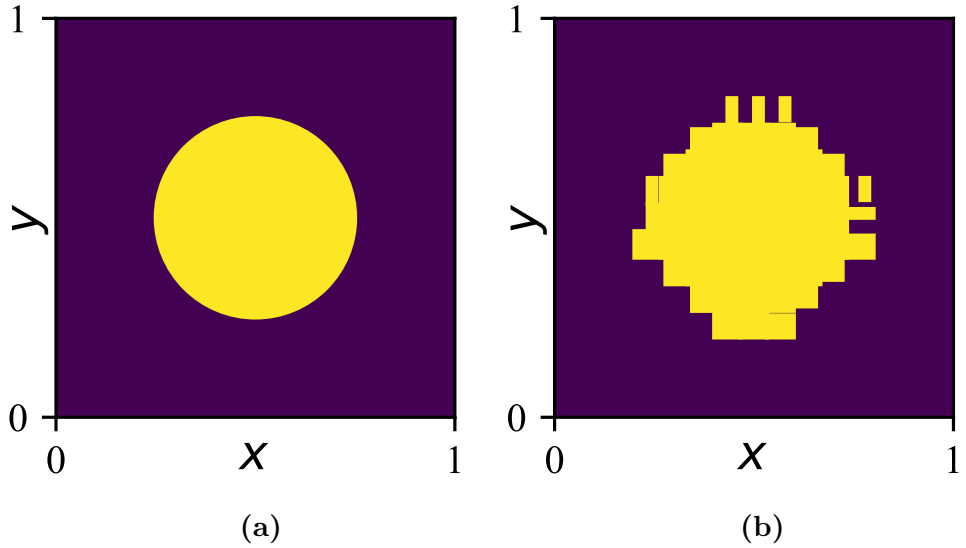


Figure 3.12: Artificial Circle problem - class distributions. Each (x, y) point is a potential input to the Fuzlearn machine, the colour of the point - its class: Purple = `False`, Yellow = `True`. **a)** Desired class distribution, the point (x, y) is `True` only if it is ≤ 0.25 away from the space center, $(0.5, 0.5)$. **b)** Fuzlearn machine converged class distribution. The training set consisted of a 45×45 grid of equispaced points. Number of subdivisions per dimension $K = 15$. The artefacts and asymmetry are present due to the training points' order.

Results and discussion

A Fuzlearn machine was initialized with $K = 15$ subdivisions, resulting in a total of 225 cells ($15^2 = 225$), to achieve an accuracy of over 90%. The training batch consisted of a dense 45×45 grid of evenly spaced points (2025 points in total) defined as follows:

$$x_{ij} = \frac{i}{44},$$
$$y_{ij} = \frac{j}{44},$$

where both i and j ranges from 0 to 44. This specific size for the training grid was chosen to ensure a significant number of points per cell (in this case, 9), while still maintaining a relatively low total number of points to avoid prolonged training sessions.

Figure 3.12 illustrates both the ideal representation of the desired class distribution and the outcome of the training process. It demonstrates how the machine has learned

to represent the class distribution using rectangles/clauses. The boundaries of the yellow rectangles are determined by the corresponding automata, as mentioned earlier in Section 3.2.

It is worth noting the presence of artefacts and the lack of symmetry in the Fuzlearn results. I believe this discrepancy can be attributed to the training order. In this case, the geometric symmetry of the actual class distribution did not align with the symmetry of the training order, which followed a row scan approach. As a result, there is a significant difference in the boundaries of the clauses between the top and bottom, as well as the left and right sides of the circle. This test emphasizes that not only the content of the training batch but also the order in which it is presented can impact the final outcome of Fuzlearn.

3.3.5 Noisy XOR 1 - Output Flip and Input Shift

One common challenge in real-world scenarios is the presence of noise. Environmental factors often introduce disturbances that attenuate the collected data. Therefore, it is crucial to investigate how Fuzlearn performs when the training data is subject to random alterations.

To conduct such a test, two artificial problems were devised, both based on the simple XOR problem. These problems aim to represent two possible scenarios in real-world settings: misclassifications of the desired class in the training data and errors, specifically value shifts, in the analogue inputs of the training data. The original XOR problem can be defined by the following equation:

$$C(x, y) = (x \geq 0.5) \wedge (y \leq 0.5) \vee (x \leq 0.5) \wedge (y \geq 0.5). \tag{3.4}$$

For each case, a set of 50 points randomly distributed in a two-dimensional space was generated for each of 500 individual training sessions. The equation 3.4 was applied to these points to assign the appropriate class. In the "Output Flip" case, with a small probability, denoted as $P(\text{flip})$, the output class of a training point is flipped to the opposite one. In the "Input Shift" case, each element of the input vector is shifted by a random value

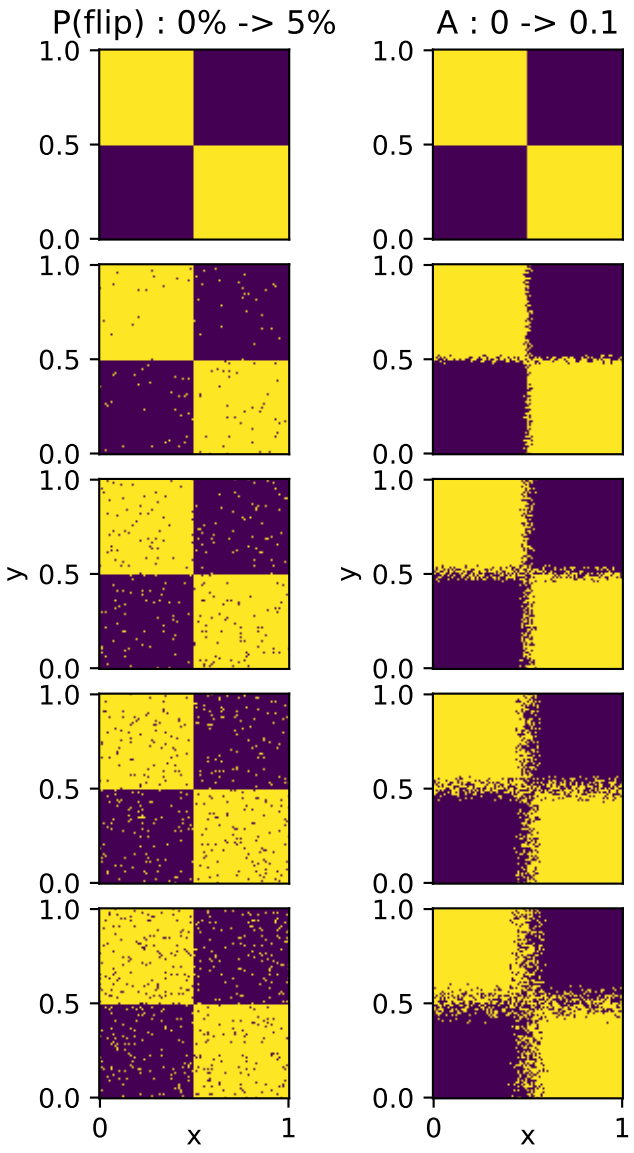


Figure 3.13: Exemplar class distributions of Noisy XOR problems: input shift and output flip. **Left:** Desired class distribution with varying probability of the output flip. Each (x, y) point adheres to the standard XOR definition, but, when fed to Fuzlearn, the desired class has a probability $P\,(\text{flip})$, to be flipped to the opposite. **Right:** Desired class distribution with varying amplitude of the input shift. Each (x, y) point adheres to the standard XOR definition, but, when fed to Fuzlearn, the original input point is shifted by adding a random number from $[-A, A]$ to each component.

from a uniform distribution $[-A, A]$, where A represents the shift amplitude. Figure 3.13 provides a visual representation of how the training sets might appear for different values of $P(\text{flip})$ and A, offering insight into their characteristics.

After a training example is processed by the machine, its accuracy is assessed by

evaluating it against a set of 1000 randomly selected unaltered points that adhere to equation 3.4. This approach reduces the likelihood of random chance significantly influencing or skewing the accuracy measurement.

Results and discussion

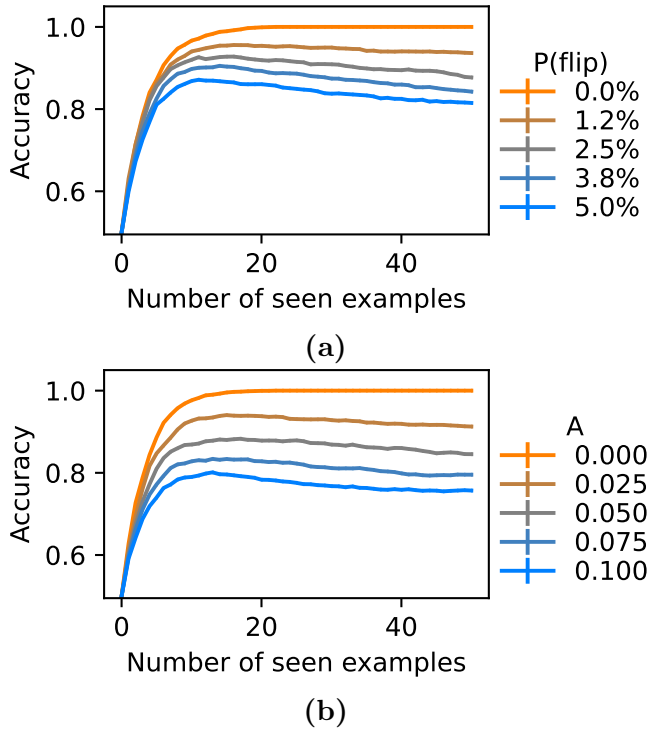


Figure 3.14: Fuzlearn machine training on the Noisy XOR problems. Number of subdivisions per dimension $K = 2$. Graphs show averaged accuracies as a function of the number of seen training points. The data is averaged over 500 individual training sessions. **a)** Output class flip variant of Noisy XOR. $P\,(\text{flip})$ - probability of desired output class of a training datapoint being altered. **b)** Input shift variant of Noisy XOR. The input is altered by adding a random number from $[-A, A]$ to each component.

Figure 3.14 illustrates the dynamics of Fuzlearn's expected accuracy for both problems. In each case, the Fuzlearn machine was initiated with $K = 2$ subdivisions per input dimension.

As anticipated, the results demonstrate that higher values of $P(\text{flip})$ or A lead to lower final accuracy. However, what's interesting is that the machine tends to achieve its peak accuracy early in the training process and then gradually becomes less accurate as more training examples are provided. This phenomenon is likely due to the influence

of erroneous training points on the clause boundaries. As the training progresses, more errors accumulate, eventually reaching a "steady state." This observation highlights a concern: a single erroneous point can potentially disrupt the learning accomplished by seeing numerous other points. This aspect makes Fuzlearn comparatively less stable than the Tsetlin Machine, which assigns greater weight to a larger number of similar points. Nonetheless, it is worth noting that the accuracies do not decrease to 50%. This indicates that despite its diminishing performance, Fuzlearn still maintains a level of usefulness and surpasses a 'coin flip' classifier.

The key insight learned from this test is that the presence of noise in the training data can have a substantial impact on the performance of Fuzlearn. Consequently, it is crucial to consider this factor when employing Fuzlearn in future applications.

3.3.6 Noisy XOR 2 - Extraneous dimensions

To get an idea of how the dimensionality of the problem limits Fuzlearn's performance, I devised this problem. As such, this is a better way of explaining why Fuzlearn was not successful against the Ozone dataset due to direct control of the core feature of that problem - the number of dimensions.

This problem is very similar to the previous two. It involves employing standard XOR training data, but without introducing any random alterations to the meaningful data. Instead, the noise arises from the inclusion of additional randomised elements or attributes in each input vector, thereby augmenting the problem's dimensionality. The intention behind conducting this test is to acquire a more controlled and comprehensive understanding of the challenges encountered during the testing with the Ozone level dataset. The aim is to determine whether the learning difficulties stemmed from the increased dimensionality of the problem or if they were due to this particular dataset being inherently difficult to solve.

The training and testing protocol follows a similar approach as in the previous XOR cases. However, in this instance, the number of training examples was set to 500.

Results and discussion

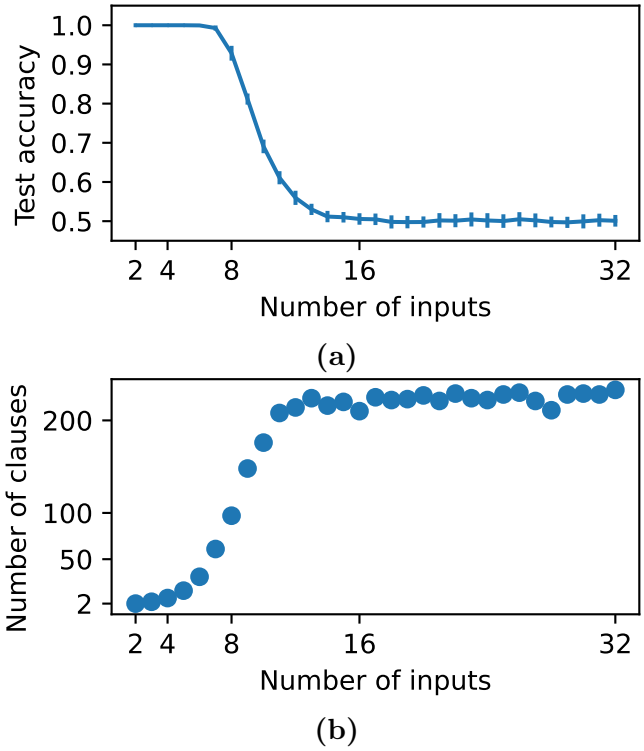


Figure 3.15: Fuzlearn machine training on the Noisy XOR problem - extended input variant. Each input point adheres to the standard XOR class distribution, but, the total number of dimensions/inputs N is increased by adding extra input elements filled with random values. Number of subdivisions per input dimension $K = 2$. Graphs show data averaged over 500 individual training. **a)** Final test accuracy vs the total number of inputs. Error bars represent the standard deviation of all 500 individual sessions. **b)** Final number of employed clauses vs the number of inputs.

Figure 3.15 displays the final expected accuracy (**a**) and the number of utilized clauses (**b**) plotted against the size of the input elements, denoted as N, after processing the entire training set containing 500 datapoints. The findings indicate that with an increasing number of inputs, the test accuracy diminishes following an equivalent amount of training. Furthermore, the results demonstrate that the final number of employed clauses is again restricted from above, with the limit being the expected number of True points in a training set, calculated as $0.5 \times 500 = 250$. This observation affirms that the challenge encountered with the Ozone dataset was not specific to the dataset itself, but rather resulted from its high dimensionality.

This presents a significant challenge as it imposes a substantial requirement on the

resources available to Fuzlearn, which grows exponentially, to be precise. However, I believe that this requirement is not inherent to Fuzlearn itself, specifically in terms of the concept of learning boundaries for analogue-to-Boolean conversion. Instead, it stems from the specific operating protocol that was employed. It is evident that if the overall prevalence of the problem is denoted as p, indicating the probability of each point in the input space being a True point, then the optimal number of clauses under this operating protocol would approach asymptotically pK^N as the value of $K \to \infty$. Hence, this issue can potentially be mitigated by devising a more efficient operating protocol. However, I fully acknowledge that accomplishing this task may prove to be exceedingly challenging.

3.4 Conclusion

This chapter introduced Fuzlearn, a novel machine learning model that aims to be a more suitable choice for in-situ ML compared to other prominent models such as neural networks and Tsetlin Machines. Through testing it on real-world problems, I have demonstrated the model's capability to tackle various challenges. However, I have also highlighted potential drawbacks of Fuzlearn that may arise in practical applications.

Specifically, I have shown that Fuzlearn successfully solves both simple problems like the Iris flower and more complex ones like HTRU2. However, the analysis of the Ozone dataset has revealed a potential issue with the resource requirements of Fuzlearn. This issue was further investigated and pinpointed in the extended input XOR problem. Additionally, the noise-induced XOR problems have indicated that Fuzlearn may face challenges when the training set is influenced by environmental factors. Nevertheless, I firmly believe that these problems are not inherent to the fundamental concept of Fuzlearn itself, but rather stem from the specific choice of the operating protocol, which can be updated as discussed in the respective section. Therefore, further exploration into the hardware implementation of Fuzlearn would not be a futile endeavour, as the operating protocol can be considered a set of rules rather than a fixed hardware/circuit configuration.

Chapter 4

Filament formation Perovskite memristors

4.1 Introduction

Incorporating Perovskite memristors into the Fuzlearn hardware implementation design could be advantageous due to their unique non-volatile properties that enable in-memory computing [25]. Here, I am interested in filament formation Perovskite memristors - memristive devices in which the resistive switching is realised via the formation of conductive paths or filaments between the device's contacts [85]. As such, I propose that Fuzlearn Automata boundaries can be stored as resistances of Halide Perovskite memristors. However, before proceeding with hardware design, it is crucial to investigate the specifics of Perovskite memristors' behaviour and create a model for use in the design process. To achieve this goal, two critical tasks must be accomplished: identification of the most suitable device configuration for the Perovskite memristors, and creation of a model that replicates the behaviour of the Perovskite memristor well enough for circuit simulation purposes. With an effective model in place, Fuzlearn circuit design can proceed with Perovskite memristors as an effective building block.

4.2 Investigating the nature of the filament

In this section, I explore the mechanism behind the filament formation in the Halide Perovskite memristors. As introduced in Chapter 2, there are two recognised interdependent mechanisms for resistive switching in Halide Perovskite materials: ionic accumulation [78] and filament formation [85, 86]. To this end, two experiments were conducted. The first experiment aimed to distinguish between the filament formation and ionic accumulation mechanisms by utilizing two differing contact metals, Silver and Gold, respectively. The objective was to determine if there were any fundamental differences in the dynamics or shape of resistive switching between these two mechanisms.

For my second experiment, I wanted to test the hypothesis that the chemical reaction between Silver and Perovskite is what causes filament formation [84]. I repeated the first experiment, but this time I added an insulating layer between the metal contact and the Perovskite to inhibit any chemical reaction happening between the materials. The purpose of this was to see if adding an insulating layer would prevent filament formation and instead allow only for ionic accumulation to happen. If this were to happen, it would support the idea that chemical reaction triggers filament formation.

As additional rationale to these experiments, I required full characterisation data to create a usable model of filament formation in Perovskite memristors. While the scientific literature provided valuable insight into the mechanisms and provided a starting point, it did not include the actual dynamic properties of the devices. The majority of the data presented was a simple current-voltage curve without any time dimension, which is insufficient for modelling purposes. To obtain the necessary temporal characteristics of resistive switching, I fabricated a series of Perovskite memristors with various setups. In summary, the conducted experiments aimed to determine the optimal configuration for my memristors, and examine how changes in device materials affected memristance.

4.2.1 Differentiating between ionic accumulation and filament formation

In this subsection, I investigate how the choice of the metal contact, namely Silver or Gold, dictates the nature of the resistive switching in Halide Perovskite memristors. In particular, I expect to see different hysteresis dynamics in the JV curves between the metals.

To achieve this, I created and characterised two sets of memristors - one with Silver contact and the other with Gold contact. For this experiment, I adopted a simple device configuration - a vertical stack of layers comprising the memristive device, see Figure 4.1. The stack in question consists of three main layers. The bottom layer is a transparent conducting oxide [100], namely Fluorine-doped Tin Oxide (FTO), which is commonly used in Perovskite solar cells [101]. FTO has various advantages, including its commercial availability and low resistivity [102], making it an ideal choice for making device contacts. Additionally, FTO-Glass slides can be easily patterned with a laser, allowing for intricate electrode patterns to be created in a simple, single-stage process without the need for any chemicals [103]. The middle layer is the Perovskite active layer, with Methylammonium Lead Iodide (MAPI) being the chosen Perovskite [104] due to its increased Iodine mobility [105]. The final layer in the stack is the metal contact, which plays a significant role in determining the kind of memristance that occurs in the final device. As previously mentioned, Silver is the control metal as it enables filament formation due to its high chemical activity. On the other hand, Gold only allows for ionic accumulation near interfaces, in theory presenting different shapes of hysteresis. The detailed fabrication procedure is presented in Chapter 6.

Results and discussion

The fabricated devices were characterised on the custom-made instrumental kit, as described in Chapter 6. The characterisation results are presented in Figure 4.2. The characterization results indicate that there is a distinct difference between the devices in their current-voltage curves. The Silver devices appear more like resistors as they exhibit

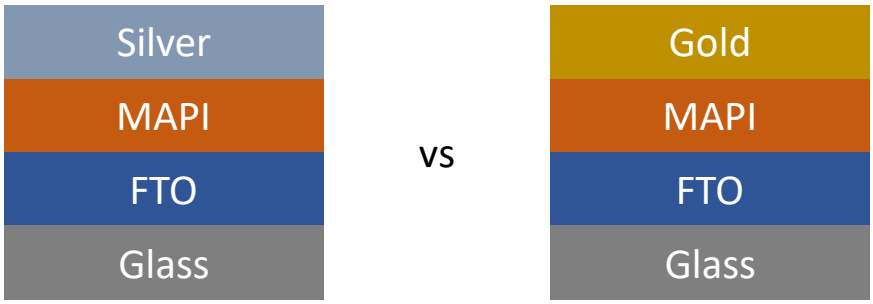


Figure 4.1: Schematic representation of the fabricated devices for the metal contact experiment. Devices - vertical stacks of layered materials. Both are comprised of a Glass-FTO-MAPI-Metal structure, where metal is either gold (Au) or silver (Ag). FTO - transparent conductive oxide (Fluorine doped Tin Oxide), MAPI - active Perovskite material.

a linear trend near the $V = 0$ region, i.e. $J \sim V$, whereas the Gold devices remain flat around $V = 0$, which is more characteristic of a diode, i.e. $J \sim \exp(V) - 1$ [106]. The latter makes sense since the devices consist of a semiconductor (MAPI) and a metal (Gold) in contact, which is expected to produce a Schottky barrier [107]. However, the fact that Silver makes an Ohmic contact can mean two things: either there is a continuous conductive path between the contacts, or there is an energy band alignment between Silver and MAPI [108].

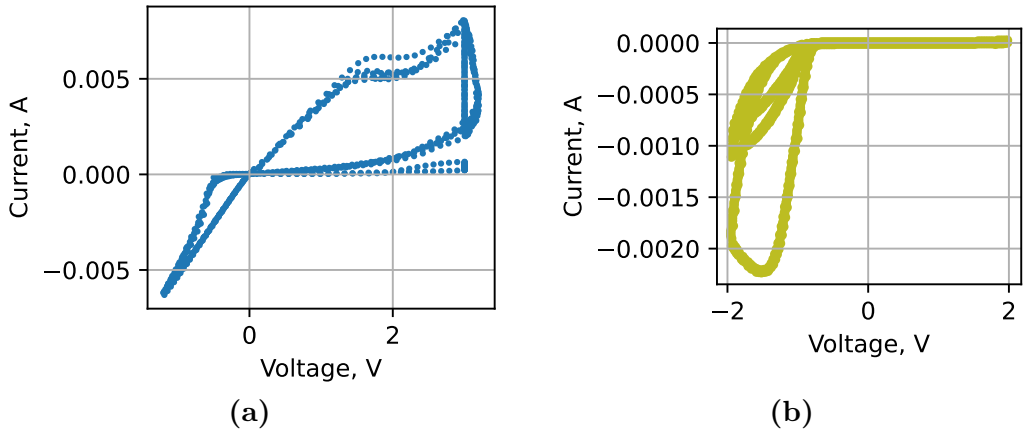


Figure 4.2: Exemplar current-voltage characteristics (JV curves) of **a)** the silver-based, and **b)** gold-based devices. The applied bias was a piece-wise linear cyclic scan. Both types of devices exhibit hysteresis, but silver ones exhibit resistor-like behaviour, i.e. $I \sim V$ around $V = 0$, compared to gold ones' diode-like behaviour, i.e. $I \sim \exp(V)$ around $V = 0$.

An important point to note is that filament growth occurs when the reverse bias is

applied to the memristors. Taking into account the polarity of the characterisation setup, which can be seen in Figure 6.6, the filament growth happens when a negative voltage is applied to the silver contact and a positive is applied to the bottom FTO contact. In such a case, the electric field drives all negatively charged particles away from the top contact. This means that during the filament growth, the region of the Perovskite layer immediately adjacent to the top contact is depleted of the Halide anions. In the case of MAPI, this means Iodine anions, I^-. This can potentially be interpreted as the filament growth being mediated by the Iodine vacancies, V_I^+, which has the opposite sign to the anion itself [109].

Additionally, while not presented in the results, it is noteworthy that Silver-based devices had a shorter lifespan and stopped being memristive after a day or two, whereas Gold-based devices exhibited memristive properties for much longer. Similar observations have been made in other materials adjacent to Perovskite [110]. It is speculated that the shortened lifespan of the Silver-based device may be due to the increased chemical reactivity of Silver, whereas Gold is more inert and allows for longer usage.

In summary, after conducting this experiment, it has been confirmed that there are two distinct resistive switching shapes achievable in Perovskites depending on the contact material. The distinction is observable, however, it remains to be proven whether they are intrinsically different phenomena or merely a representation of the same thing in disguise. As of now, it can only be affirmed that Silver produces memristors with a more pronounced, resistor-like memristance, while Gold ones are more diode-like.

4.2.2 Controlling filament formation

With the apparent difference in resistive switching between the metals being shown in the previous experiment, I am narrowing down the previous hypothesis. In this subsection, I am aiming to confirm that it is Silver that enables the filament formation in Halide Perovskite, specifically a chemical reaction between Silver and Perovskite.

To achieve this, I placed a thin layer of chemically insulating material, namely Polymethyl Methacrylate (PMMA) [111], between Silver and MAPI, which can inhibit any

possible chemical reaction at the interface. The hypothesis is that this will cause the resistive switching to significantly diminish, i.e. reducing the size of the hysteresis, indicating that filament formation is indeed activated via the Silver-MAPI chemical reaction. If this is correct then the overall JV curve should be more akin to that of Gold memristors because in this case, the only allowed resistive switching mechanism is ionic accumulation. However, if the JV curve remains largely the same, with a large hysteresis and linear behaviour, it would suggest that something else entirely is at play and the chemistry of the Silver-MAPI interface is not important.

A secondary reason for this experiment is to potentially extend the lifespan of the memristors. As mentioned in the previous subsection, Silver-based memristors were prone to fast degradation. As such, If the PMMA layer would not inhibit the memristance of Silver devices, i.e. there is no chemical reaction between Silver and MAPI, then the only available degradation pathway left is through the air - Perovskites are known to degrade under contact with airborne molecules [112]. As such the idea is that this additional layer will protect MAPI from contact with air and prolong the memristors' lifespan.

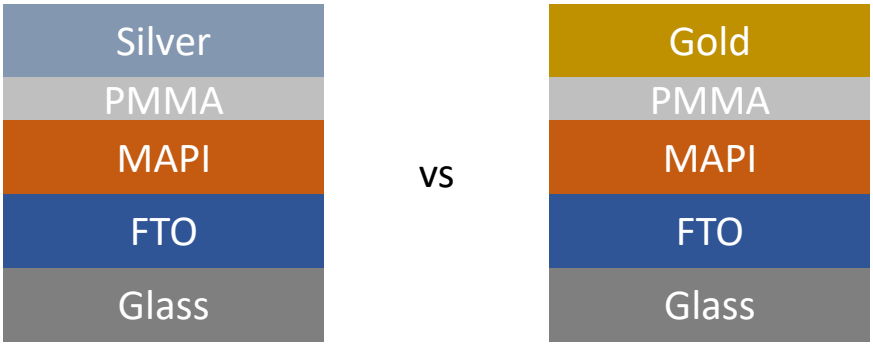


Figure 4.3: Schematic representation of the fabricated devices for the insulating layer experiment. Devices are vertical stacks of layered materials. Both are comprised of a Glass-FTO-MAPI-PMMA-Metal structure, where the metal is either gold (Au) or silver (Ag). PMMA - Polymethyl methacrylate - insulating polymer material. FTO - transparent conductive oxide (Fluorine doped Tin Oxide), MAPI - active Perovskite material.

The devices for this experiment were fabricated and characterised in a similar fashion as for the previous experiment, see Figure 4.3, with the notable difference of additional PMMA layer between MAPI and metal contact. The specifics of PMMA layer fabrication

and application can be found in Chapter 6. Both metals, Gold and Silver, were used as top contact material, thus, creating two sets of devices. While Silver was the material under investigation, Gold was utilised to create a control group, whose resistive switching shape should remain the same.

Results and discussion

In Figure 4.4, the current-voltage characteristics of Silver- and Gold-based devices with an additional thin layer of PMMA are presented, showing several typical JV curves for both types of devices. The curves under the same metal name represent devices fabricated under the same conditions. The main hypothesis of this experiment was that PMMA would inhibit filament formation and make Silver-based devices' resistive switching look more like Gold devices', and the data suggests that this is indeed correct. When comparing Silver curves with PMMA and without PMMA in Figure 4.2, it becomes more diode-like near the $V = 0$ region, which is the same as Gold's. However, there are a couple of oddities that need addressing.

Firstly, under the same fabrication condition, the memristance can be either reduced, but not removed (see Figure 4.4 bottom), or it could be completely quenched (see Figure 4.4 top). The only reasonable explanation for this is that the PMMA films contained pinholes. It was reported before that PMMA would contain pinholes if one tries to spin coat a very thin layer of it [113]. As such, it became possible for some devices to have almost no memristance and others to retain their memristive properties while both were located on the same substrate.

Secondly, the memristance was quenched by PMMA even for Gold-based memristors, which was unexpected. The only reasonable explanation for this phenomenon is that even for ionic accumulation type of memristance, there is some chemistry happening between the contact material and the Perovskite, even for such an inert metal as Gold. Some research suggests that under bias, Gold reacts with Iodine ions in the Perovskite and forms unstable Gold Iodide which naturally decays after the bias is removed [84].

Based on the results of the initial hypothesis, it appears that the secondary goal of the

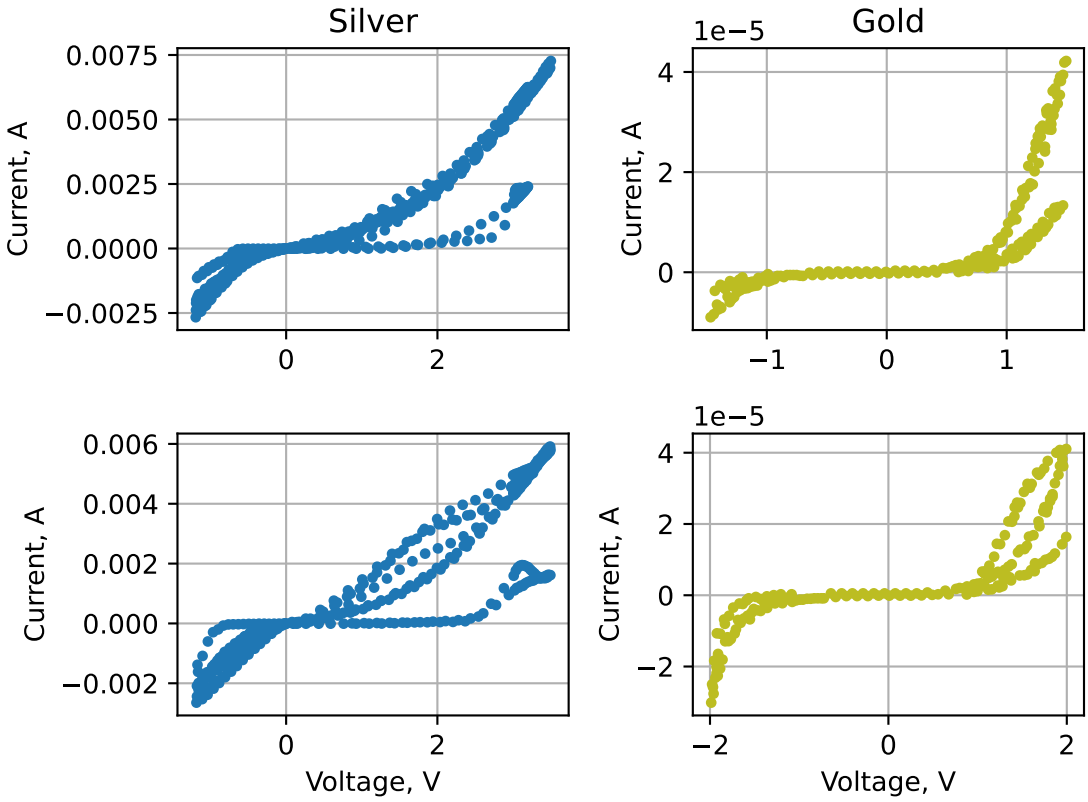


Figure 4.4: Exemplar current-voltage characterisation of the silver-based (left) and the gold-based (right) devices with added PMMA layer in between MAPI and the metal contact. Hysteresis is significantly inhibited in both cases compared to the device stack with no PMMA. Both silver- and gold-contacted devices exhibit flattening of the current around $V = 0$, signifying the inhibition of silver-induced filament formation.

experiment to use PMMA as a means of preventing degradation in Perovskite memristors is unachievable. PMMA was found to effectively reduce memristance in both metals, indicating that it cannot be utilized as a means of preventing Perovskite degradation without negatively impacting the memristor's intended function.

In conclusion for both experiments that were performed, the choice of metal contact plays a significant role in determining the type of resistive switching that is attained. Specifically, the use of Silver and Gold produced notably different results. Moreover, it was determined that the filament formation mechanism, which is associated with Silver-based memristors, is reliant on the direct contact between Silver and Perovskite. This suggests that a filament formation process is mediated via a chemical reaction.

Putting the experimental results into the perspective of the entire project, I decided that the Silver-based no PMMA MAPI memristor should be adopted as the model device for further modelling efforts. This is because their memristance is more pronounced than that of the Gold ones, i.e. they have larger and more pronounced hysteresis. Additionally, resistor-like devices are easier to utilize in circuit design than diode-like devices.

4.3 Developing a filament memristor model

To incorporate Perovsktie memristors from the previous section into hardware design and circuit simulation, a model needs to be developed that can be integrated into circuit simulation software. However, this poses a limitation on the complexity of the model, as the circuit will inevitably use multiple memristors at the same time, along with other electrical components. To ensure an effective circuit design process, there should be a short latency between designing and simulating the circuit, which means the model should not take long to simulate.

To address this challenge, I first briefly review the existing memristor models, along with their underlying workings. In addition, for each of the models I present an argument on why they are not well-suited for use in my project. Therefore, after the review, I present a detailed description of my model.

4.3.1 Existing memristor models

The starting point for memristors is often traced back to the work of Chua [68], who introduced a mathematical framework for creating models of these devices. While Chua's work is highly influential, it is important to note that he did not create a concrete model of a memristor. Rather, his framework provides a useful tool for other researchers to develop their own models.

After the first-ever realisation of the memristor in Titania [114], Pickett proposed his rather influential model [115]. In his model, he suggested that the device has a conductor-insulator-conductor structure, and, thus, the memristance is achieved through the mod-

ulation of the electroforming channel width, i.e. changing the size of the insulating gap. As such, this model is based on the Simons model of barrier tunnelling [116]. This is a justifiable choice since Titania (TiO_2) is a known insulating material. However, Halide Perovskites are not insulators, especially MAPI which I used for my devices. Incidentally, quantum tunnelling might not be the most appropriate theoretical framework to describe the flow of the current through my devices. Therefore, I don't think this model is well-suited to model my Silver-based Halide Perovskite memristors.

Another model, inspired partially by Pickett's model, was presented by Gupta [117]. This model is of particular interest as it directly considers Halide Perovskites as the base material. In this model, memristance is achieved through the modulation of the state variable. However, unlike Pickett's model where it controls the gap width, here, it determines the type of contact that Perovskite makes with the contact material: Ohmic contact or Schottky barrier. Although this model was applied to Halide Perovskite memristors, they were actually Gold-based memristors. As such, the J-V characteristics of their devices were not similar to those of my Silver-based devices. Therefore, I was not confident in adopting it to represent my memristors.

While there could already be other more comprehensive memristor models, this list reviews only those which were published before I started working on my model. As such, here, I do not consider any model which was published after 2021.

4.3.2 Outline of my model

The idea behind my model is that the filament (or filaments) can be considered as cylindrical structures that grow between the contacts inside the bulk Perovskite. As the filament grows, it replaces the bulk Perovskite material with itself. The filament's material is considered to be much more conductive than the bulk's material, which results in the lowering of the total resistance of the device. The schematic representation of this idea is presented in Figure 4.5.

The microscopic nature of the filament is still elusive. Whether it is made of Silver atoms/ions [85], or ionic vacancies [84] remains unclear. The conductance could be in-

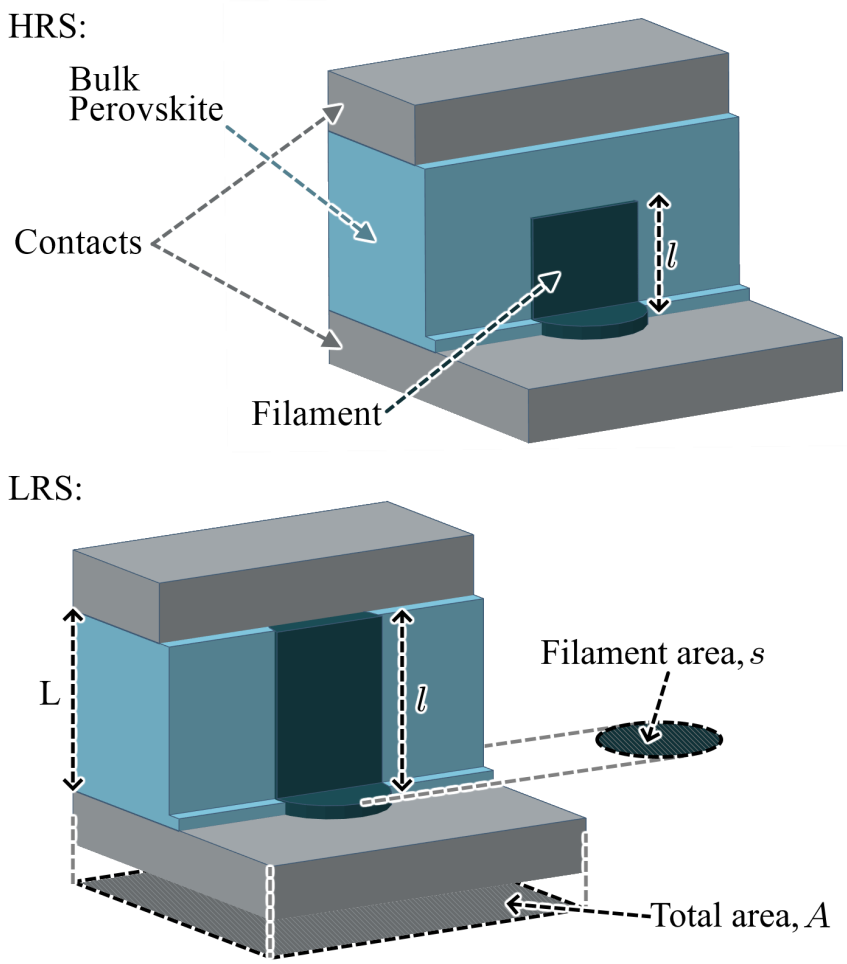


Figure 4.5: Geometric representation of my filament formation model. The device is comprised of Perovskite active material placed in between two conductive contact layers. The filament is a cylindrical structure growing from one contact to the other. The resistance decreases with filament growth due to the filament's more conductive material. LRS - Low resistive state, the filament is complete, and total resistance is at its lowest. HRS - High resistive state, the filament is not yet full, and total resistance is high.

creased due to the inherent conductivity of the filament, or the filament could be modifying the surrounding bulk material, e.g. via doping. To bypass this uncertainty, my model is designed such that it disregards the microscopic nature of the filament, abstracting it away with a material mobility parameter. Therefore, my model can be applied to any possible case, demonstrating its versatility and general applicability.

It is crucial to understand that due to the simple nature of my model, it is impossible to determine if each memristor contains only one filament or if there is a group of filaments

that collectively affect the device's total resistance. Hence, it can be stated that the filament area, s, as depicted in Figure 4.5, represents the combined area of all filaments if there is more than one.

In the following subsections, I provide a more detailed description of certain aspects of my model. First, I explain how the filament affects the device's resistance. Then, I discuss the dynamical component of filament growth and its relation to external bias. It's worth noting that the model's design compromises accuracy by relying on classical laws of electrodynamics. However, it is important to note that accuracy is not the model's primary objective. Instead, it is designed to provide sufficient accuracy for circuit simulation applications, which will enable designing hardware for Fuzlearn architecture.

4.3.3 Dependence of the device's resistance on the filament geometry

The first important point that needs to be defined is how the filament affects the total resistance of the device. To connect the geometry of my model with the macroscopic observation of a device measurement, I define how the memristor's total resistance changes with the filaments' growth using Ohm's law [118].

As seen in Figure 4.5, the entire device can be subdivided into two regions. The first region is comprised of all points of the bulk that do not lie directly above the filament area, s. That is all the points of the bulk represented in the light-blue colour in the low resistive state. According to the region's definition, no filament grows in the first region, resulting in a constant resistance regardless of voltage or time. This resistance is known as the static resistance, denoted as R_S, and can be calculated using the following equation:

$$R_S = \rho_0 \frac{L}{A - s}. \tag{4.1}$$

In the second region, filaments progressively grow and replace the bulk perovskite material under the applied bias. Consequently, the resistance in this region decreases due to filament growth. I model this variable resistance as a series connection between the filament and the unconverted region [119]. The expression for this variable resistance, denoted as R_V,

is as follows:

$$R_V = \rho_0 \frac{L(1-z)}{s} + \rho_1 \frac{Lz}{s}. \tag{4.2}$$

In the equation above, the first term represents the resistance of the unconverted bulk and the second term is the resistance of the filament. In both eq. 4.1 and eq. 4.2, ρ_0 and ρ_1 are the resistivities of the bulk and the filament material respectively. The filament factor, $z = l/L$, represents the filament length as the fraction of the device thickness. Therefore, $z = 0$ indicates a complete absence of a filament, i.e. the device is in its high resistive state, while $z = 1$ indicates a fully formed filament that connects the opposite electrodes, i.e. low resistive state.

The total resistance of the memristor is then defined as the parallel connection [119] of the variable resistance and the static resistance. By combining eq. 4.1 and eq. 4.2, the total resistance, denoted as R_T, can be expressed as

$$\frac{1}{R_T} = \frac{1}{R_S} + \frac{1}{R_V}. \tag{4.3}$$

As this equation suggests, my model assumes that the electrical current flows only in the direction parallel to the growth of the filament, and there is no perpendicular component between the first and second regions of the device. I think that this assumption is rather simplistic and, as such, not entirely accurate. A more accurate approach, perhaps, would involve solving a system of differential equations including Maxwell's [120], Ohm's, Drift-Diffusion [121], etc. However, correcting it would require a substantial increase in the computational complexity of the model, rendering it ineffective for circuit design [122].

4.3.4 Filament growth's dependence on applied bias

The second necessary component of my model is the connection between the filament growth and the applied bias. In this subsection, I present the relation between the filament factor, z, and the level of the applied bias, V.

In my model, I propose that the filament's growth is akin to the movement of charged

particles in the media [123]. As such, it is governed by the drift velocity expression:

$$v = \mu E, \tag{4.4}$$

where v represents the velocity at which the front/tip of the filament moves towards the opposing electrode ($v = \dot{l}$). The value of μ represents the mobility of the filament front, which could be linked to the mobility of ionic vacancies in the Perovskite material. After all, these vacancies are the probable mediators of the filament's growth [84]. However, it should be noted that the relationship may not be direct, and further research is required to understand and prove its form. Finally, E indicates the electric field strength. Based on these considerations, we can assert that

$$v = \dot{l} = (\dot{Lz}) = L\dot{z}. \tag{4.5}$$

Assuming that the contacts are sufficiently conductive to be equipotential and that the entire applied voltage drops linearly across the bulk Perovskite layer, we can state that

$$E = \frac{V}{L}. \tag{4.6}$$

By combining all of the above equations, we arrive at

$$\dot{z} = \frac{\mu}{L^2} V. \tag{4.7}$$

In summary, eq. 4.7 presenters the relation between the speed of the filament's growth and the value of the applied bias. Solving this differential equation for z and plugging it back to the total resistance eq. 4.2 should yield the theoretical resistance of the entire device. However, there are still a few concerns related to this equation that should be addressed before proceeding to the actual model simulation.

Asymmetry in filament growth/retraction

After observing the JV curve of a real filament formation memristor in Figure 4.2, it became clear that eq. 4.7, which was supposed to describe the connection between the filament factor, z, and the applied bias, V, was not able to accurately represent the current-voltage characteristic of the real Perovskite memristor. The actual device demonstrated a significant level of asymmetry in its resistance-switching behaviour. Specifically, the shapes of the loops on the negative and positive sides have substantially different sizes. To address this, a straightforward correction was introduced to the eq. 4.7, resulting in

$$\dot{z} = \begin{cases} \frac{\mu_0}{L^2} V & \text{, if } V \geq 0, \\ \frac{\mu_1}{L^2} V & \text{, if } V < 0. \end{cases} \tag{4.8}$$

This expression now separates the filament front's mobility, μ, into two cases according to the applied bias's direction. This adds to the model a necessary capability for asymmetry. A further generalisation of this concept can be expressed as

$$\dot{z} = \frac{\hat{\mu}\left(V\right)}{L^2} V, \tag{4.9}$$

where $\hat{\mu}\left(V\right)$ is the filament front's mobility as a function of the applied voltage. In this form, it can accommodate a more intricate relation between the filament and the applied bias. However, at this point hin time, there is not enough evidence or data for any specific form of this function. Thus, eq. 4.8 is considered to be adequate.

Geometric constraints of the filament

In addition to addressing the asymmetry behaviour, another necessary modification is required in my model. One of the logical constraints is that the filament must be geometrically limited within the Perovskite active layer. This means that the filament factor, denoted as z, must be within the range from 0 to 1. As a result, another update to the filament-bias relation, as seen in equation 4.10, is necessary. In this equation, $\dot{z}'$ represents the actual rate of change of the filament factor, whereas $\dot{z}$ represents the preliminary rate

of change as stated in equation 4.8, which does not take into account the range constraint.

$$\dot{z}' = \begin{cases} \dot{z} & \text{, if } 0 < z < 1, \\ \max(\dot{z}, 0) & \text{, if } z = 0, \\ \min(\dot{z}, 0) & \text{, if } z = 1, \end{cases} \tag{4.10}$$

This modification ensures that the value of z never goes above 1 or below 0, preventing instances of nonsensical resistance values.

It's important to note that the last two modifications transformed my model into what's called a *hybrid system* [124]. In general, a hybrid system is a dynamic system with both continuous and discrete states/state changes. In the case of my model, it has discrete jumps in the rate of change of the filament factor at the boundaries of z as well as at the sign change of V. This makes it challenging to simulate the system. To address this issue and improve the reproducibility and accuracy of my simulations, I use extended accuracy settings in all of my software implementations where applicable.

4.4 Model Verification

Before the model could be used for circuit simulation, it was necessary to ensure that it could accurately replicate the overall behaviour of the real-world memristor. To achieve this, I opted to fit the model into the experimental data through parameter optimization. If it successfully fit, then it was deemed appropriate for use; however, if it did not, it meant that the model needed further refinement. To fit the model into the data, it needed to be implemented in software. But before translating the model to software, there were some concerns that needed to be addressed.

Firstly, the model was defined in terms of microscopic parameters such as device thickness, mobilities, resistivities, etc. These parameters cannot be discerned from the data directly, only through numerical optimisation. In addition, these parameters were interconnected in the model equations to such an extent that a numerical method would not yield any useful results. To resolve this, it was necessary to redefine and recombine the

model's parameters into a new set of parameters that represent the device's macroscopic behaviour rather than microscopic. With the new set of model parameters, numerical fitting could take place.

The second concern to be addressed was the computational complexity and potential instability due to the hybrid nature of the model. To better control the verification process and avoid any errors/bugs, the verification process was subdivided into two logical modules: model simulation and parameter optimization. This way, the simulation module is unconcerned about where it would be utilized later on. Its only concern is to give a stable simulation of the model behaviour given a specific set of parameters. Parameter optimization would be only concerned about optimizing the parameter set via comparison of the model's simulated behaviour with the real data, without any care for the nature of the underlying model that is being simulated. This is known as the *separation of concerns* [125].

4.4.1 Parameter simplification

Before fitting the model into the data, its parameter set must be converted to make it suitable for fitting. One of the reasons for this is that certain parameters within the model are coupled within the governing equations, see eq. 4.7 and eq. 4.8. Filament front mobility, μ, and device thickness, L, are two such parameters. In the process of parameter optimization, a change in one of these parameters can be re-expressed as a corresponding change in the other, making these two parameters redundant/degenerate [126]. This redundancy has the potential to halt the fitting process. To tackle this issue, a possible solution is to aggregate the mobility and thickness parameters into a single parameter, $k = \mu/L$. The units of this combined parameter now become $[V]^{-1}[s]^{-1}$, which hints at its physical meaning - the rate of change of z per every unit of the applied bias. By applying this amalgamation procedure to eq. 4.8, it transforms into

$$\dot{z} = \begin{cases} k_0 V & \text{, if } V \geq 0, \\ k_1 V & \text{, if } V < 0. \end{cases} \tag{4.11}$$

In this compressed form, the parameter redundancy is removed, as k_0 and k_1 are independent. In other words, a change in one of them can never produce the same model behaviour as a change in the other.

Another reason for parameter recombination is that it can speed up the fitting process. By combining model parameters in a certain way, new parameters can be obtained directly from the JV curves. This can eliminate the need for computationally intensive numerical fitting. For instance, it is impossible to extract filament area, s, and material resistivities, ρ_0 and ρ_1, from experimental data. However, the device's bulk resistance, R_T, can be directly extracted from a JV curve. As such, if the model is redefined in terms of bulk macroscopic parameters, the fitting process is accelerated substantially.

However, another important point to note is that during the characterisation of the memristors the current was measured while the voltage was controlled. This means that the primary source of uncertainty in the measurement lies within the current measurement. As such, a more suitable parameter for extraction from a JV curve is the conductance, G, of the device rather than the resistance, R. In conductance, $G = J/V$, the uncertainty range is linearly proportional to the current's uncertainty, while in resistance, $R = V/J$, it is inversely proportional, producing poles and singularities. Therefore, for the purpose of model fitting, G serves as a more stable metric than R. The device's total conductance can be expressed as

$$G_T = \frac{1}{R_T} = \frac{1}{R_S} + \frac{1}{R_V}. \tag{4.12}$$

Following up on the fact that s, ρ_0 and ρ_1 are not directly measurable, both the static and variable regions have to be represented with a different set of parameters. As such, the static conductance emerges as one of the parameters in the new recombined set

$$\frac{1}{R_S} = G_S. \tag{4.13}$$

Applying the transition from resistance to conductance onto eq. 4.2, it takes a new form

$$R_V = \frac{(1-z)}{G_{V0}} + \frac{z}{G_{V1}}, \tag{4.14}$$

where the conductances G_{V0} and G_{V1} represent the conductance values of the variable region of the device at its extremes — when the filament is absent ($z = 0$) and when the filament is fully extended ($z = 1$). Both of these parameters are related to the original parameters as

$$G_{V0} = \frac{s}{\rho_0 L}, \tag{4.15}$$

$$G_{V1} = \frac{s}{\rho_1 L}. \tag{4.16}$$

G_{V0} and G_{V1} are included in the new set of parameters considered for the parameter fitting.

Table 4.1: Memristor model's parameters compression. **Top:** Original (blue) vs. Compressed (green) parametric sets. The original set is based on the device's microscopic properties, while the compressed set captures its macroscopic electro-dynamical behaviour. **Bottom:** The relation matrix between new and old parameters. A dot at the parameter row and column crossing represents the relation between the parameters.

$$\{\mu_0, \quad \mu_1, \quad L, \quad \rho_0, \quad \rho_1, \quad s, \quad A\}$$

VS

$$\{k_0, \quad k_1, \quad G_S, \quad G_{V0}, \quad G_{V1}\}$$

New ↓ \Old →	μ_0	μ_1	L	ρ_0	ρ_1	s	A
k_0	•		•				
k_1		•	•				
G_S			•	•		•	•
G_{V0}			•	•	•	•	•
G_{V1}			•	•	•	•	•

In summary, after combining and compressing the original set of the model parameters, the new set of the parameters is presented in Table 4.1. The overall quantity of parameters has been decreased from seven to five, which is an advantageous outcome as it makes the parameters more manageable and necessitates less computation for numerical fitting.

4.4.2 Python implementation of the model

In order to use the model in circuit simulation, it is essential to verify that it can recreate the overall behaviour of a real-world memristor. The most effective way to do this is by implementing it numerically and comparing the results of the theoretical model prediction with real-world device data. However, implementing the model in a programming language is a challenging task due to its hybrid nature. To ensure that the software verification and parameter optimization process is valid and viable, it is crucial to address any possible issues related to the hybrid nature beforehand.

To achieve this, the optimization process has been divided into two separate independent modules. The first module, the model simulation module, is responsible for simulating the model by solving the governing equations. Here, the parameters are provided externally and remain static for the entire duration of the simulation. This way, any issue regarding the hybrid nature can be caught in this module and dealt with on this level [127]. The second module is a general-purpose parameter optimization module that does not concern itself with how the model works. It simply provides the model with parameters and expects model predictions in return. This approach of subdivision in software is known as separation of concerns and is widely used in the software engineering community [125, 128].

The model was implemented using the Python programming language, primarily because of its ease of use, the existence of open-source solutions for different problems, its widespread use throughout the scientific community, and, of course, due to my personal preferences and experience with the language [92]. The Numpy and Scipy libraries have been used as the foundation for both modules. These libraries are Python libraries containing numerous functions specifically tailored for scientific numerical computing [129].

Model simulation module

The main aspect of the model simulation was to solve the differential equations that describe the evolution of the filament factor, z, w.r.t. time and applied bias. However,

this was a challenging task due to the hybrid nature of the equations, which contain discrete 'jumps' such as when z reaches one of its geometrical constraints in eq. 4.10 or when V changes sign in eq. 4.11.

To tackle these equations, Scipy's `solve_ivp` function was utilised, which is an arbitrary ODE solver. The advantage of using this solver is that it can solve any given set of differential equations of any kind using the Explicit Runge-Kutta method of order 5(4) [130], although alternative solving schemes are available [131].

However, the hybrid nature of the equations showed itself in the results of the ODE solver, as indicated in Figure 4.6(a). The figure presents the results of solving eq. 4.10 and eq. 4.11 for an arbitrary square wave bias ($V \in \{-1, 1\}$, $f = 10$ Hz) [132] using the default settings of `solve_ivp`. It can be observed from the figure that the geometrical constraints of eq. 4.10 are not always met as the value of z can go outside of the $[0, 1]$ range. Additionally, for a periodic bias, it is expected that the value of z should also be periodic with time, but there is a gap in the value of z in the region of $t \in [6, 8]$ s. The hypothesis for such behaviour was that `solve_ivp` could not predict when exactly a system jump happens and, as such, cannot switch to the right branches in the equations at the right time.

To amend this problem, the `max_step` parameter of `solve_ivp` was utilised, which controls the maximum time step of the solver [131]. It was set to half of the time difference between consecutive points of the bias. The results of this modification can be seen in Figure 4.6(b). With such modification in place, the value of z is now properly constrained within the $[0, 1]$ range, and it is periodic with respect to time as expected. As such, for all future model simulations, the `max_step` parameter was set to half of the minimum time difference between two points in the corresponding experimental dataset.

One evident disadvantage of using `solve_ivp` and setting its `max_step` parameter is that it extends the simulation duration. However, within the scope of the project, it was an acceptable compromise since the duration of most simulations was not longer than 3 seconds. Ideally, an analytical approach could be utilised to predict every discrete jump in the system exactly. However, it would not have been practical for the purposes of

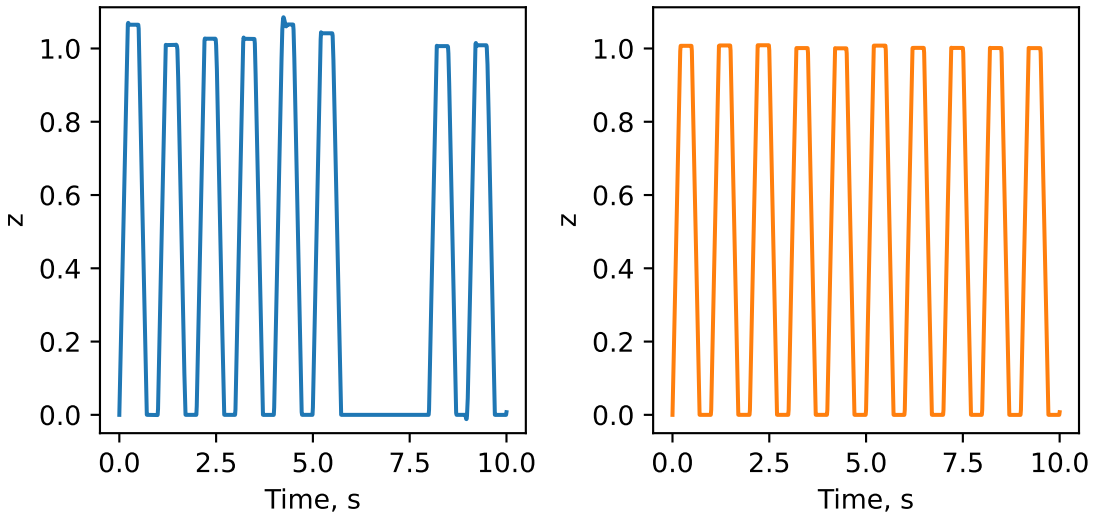


Figure 4.6: Numerical solution of filament factor equation 4.11 with a square wave bias signal. **a)** No `max_step` case. The geometrical constraints imposed on the filament, i.e. $0 \leq z \leq 1$, are not satisfied. Periodicity is broken around $t = 7$s. **b)** With `max_step`. The filament factor value is constrained to $[0, 1]$ as it should, i.e. the filament is always located within the device. The filament factor is periodic.

the overall project since it would have required an extensive modification of the circuit simulation software to accommodate the model.

To conclude the model simulation, after the equations for z are solved, its value over time is incorporated into eq. 4.14 with the rest of the model parameters to obtain the value of the variable conductance. This, in turn, is incorporated into the total conductance, G_T, see eq. 4.12. The total device conductance is then used in $J = G_T V$ to obtain the theoretical prediction of the current through the memristor for a given bias V. This current prediction is the final result of the model simulation module which is then given to the parameter fitting module for comparison with the experimental data.

Parameter fitting module

The parameter fitting module involves adjusting the parameters of the model to fit the model's predictions into the experimental data. As mentioned above, the model's parameter set was redefined in terms of device macroscopic level parameters, making it easy to

extract some of the parameters directly from the JV curve without the need for a numerical optimization algorithm. Specifically, the highest conductance that the device could attain, G_H, was directly taken from the JV data. It was assumed that when the total conductance is at its highest, the filament is fully extended, i.e. $G_T\,(z = 1) = G_H$.

However, the lowest device conductance, G_L, was much more difficult to extract from the JV data due to the greater relative uncertainty of the current measurement for small currents. The only noteworthy aspect of G_L is that it is orders of magnitude smaller than G_H. As such, it was estimated as $G_L = G_H/100$. While it is an arbitrary choice, it has very little consequences on the overall model behaviour as long as G_L can be described as 'orders of magnitude smaller than G_H'. Using similar reasoning, the static region's conductance was estimated to be $G_S = G_L/2$.

As such, the total device conductance, G_T, equals G_H when $z = 1$ and G_L when $z = 0$. These can be directly connected to the model parameters from eq. 4.12 and eq. 4.14 as follows:

$$G_H = G_{V1} + G_S, \tag{4.17}$$

$$G_L = G_{V0} + G_S. \tag{4.18}$$

In summary, all conductance parameters of the model, namely G_{V0}, G_{V1}, and G_S were extracted directly from the experimental data.

Unlike the conductances, the values for k_0 and k_1 parameters could not be extracted so trivially. Therefore, a global optimization algorithm was utilized to find the optimal values for these parameters. The reason behind opting for a global optimization as opposed to local optimization is the fact that there was no guarantee that there were no multiple local minima for the model fit [133]. As such, Scipy's `dual_anealing` method [134] was used, which implements the Simulated Annealing algorithm [135, 136]. The algorithm is inspired by the physical process of annealing [137]. In it, each potential solution, i.e. a pair of (k_0, k_1), is associated with an energy level, equated to the combined error of the model prediction for that specific pair. Transitions between candidate solutions are

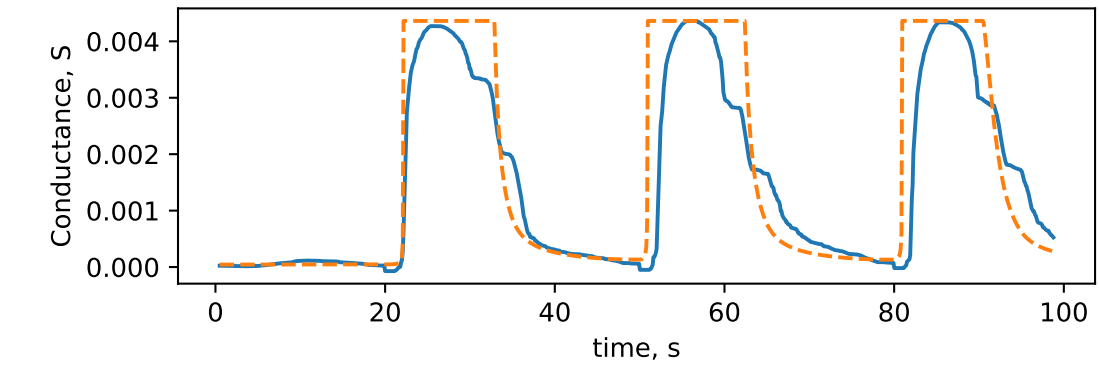
simulated as a stochastical hopping of a real physical system between its stable energy levels, favouring those transitions where the system transfers to a lower energy level. As such, the rate of the solution transition is dictated by their energy difference and the underlying 'temperature'. As the optimization progresses, the temperature gradually decreases, mimicking the cooling in the annealing process [138].

4.4.3 Results and discussion

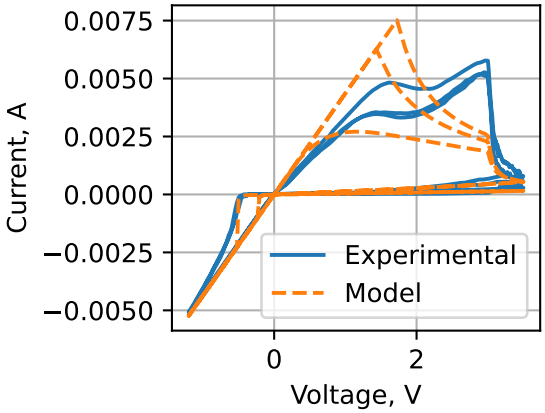
To verify the model's ability to recreate the behaviour of a real-world filament-formation Halide Perovskite memristor, its Python implementation was tested against the experimental data, collected during the experimental stage of this project. The results of the parameter fitting are presented in Figure 4.7(a). This graph shows the conductance evolution with time for the real memristor and the model's prediction. It demonstrates that the model effectively reproduces the asymmetric extension and retraction of the filament, as well as the overall timing of the resistance switching.

It is important to note that while the model captures the general behaviour, it does not replicate intricate details. When using the model's conductance to produce the appropriate JV curve, as shown in Figure 4.7(b), it becomes evident that the disparities, namely in current, are more pronounced. Discrepancies between the model and experimental data arise due to the inherent limitations of the model's accuracy, which stems from its reliance on simplified physics. For instance, it's well established that Ohm's law is not accurate enough to describe the flow of electrical current through a semiconductor [139]. Another simplification is that the model disregards the real-world complexity of the Perovskite layer, typically exhibiting a poly-crystalline structure [140].

While there could be many contributors to the discrepancy in the results, I believe that one of the most prominent is the cylindrical geometry of the filament. My conjecture suggests that transitioning from a cylindrical to a variable cross-sectional area shape, such as a conical shape, might enhance the model's accuracy. The idea of the filament having a conical or conical-like shape is not unsubstantiated, since it's been shown that filaments can have conical shapes in other filament-based memristive materials [141]. On



(a)



(b)

Parameter	Value	units
k_0	-0.00425	$[\mathrm{V}]^{-1}[\mathrm{s}]^{-1}$
k_1	-1.8325	$[\mathrm{V}]^{-1}[\mathrm{s}]^{-1}$
G_S	2.18×10^{-5}	$[\Omega]^{-1}$
G_{V0}	2.18×10^{-5}	$[\Omega]^{-1}$
G_{V1}	4.34×10^{-3}	$[\Omega]^{-1}$

(c)

Figure 4.7: Filament formation memristor model fitting results. The blue solid lines represent the original experimental data, orange dashed lines - the model's prediction. **a)** Graph of the device conductance, $G = I/V$, w.r.t. time. **b)** Current-voltage plot, reconstructed using $I = G \times V$. **c)** Table of optimised parameters.

top of that, the accuracy improvement is anticipated due to the incorporation of additional parameters, allowing the model more flexibility in fitting the data. However, within the context of this project, reshaping the filament presents challenges in terms of increased computational complexity. The translational symmetry along the filament's growth axis in a cylindrical geometry simplifies the resistance calculations of the variable region of the memristor, expressed as a simple formula shown in eq. 4.2. Contrarily, a non-uniform cross-sectional shape would necessitate solving a non-trivial integral. In some cases, closed-form solutions might be non-computable, requiring the use of numerical integration for each simulation time step, thus, increasing the simulation times and slowing down the circuit design pipeline.

As previously mentioned, for the purposes of this project, the focus of the model is not to be 100% accurate, but it is instead to have the ability to replicate the overall behaviour of a filament formation Perovskite memristors with a simple, computationally tractable, model for purposes of circuit design applications. Based on these observations, I consider this model to be moderately successful. Consequently, I am confident in proceeding to the next phase of my project, which entails transferring this model into circuit simulation software and utilizing it for the design of dedicated hardware for Fuzlearn.

4.5 Conclusion

In this chapter, I proposed to use filament formation Halide Perovskite memristors as a key component for building hardware acceleration for Fuzlearn ML architecture. To achieve this, I needed a robust and effective model of such a memristor for circuit simulation.

To create the model, I conducted a series of experiments to investigate the mechanism behind resistive switching in Halide Perovskite memristors with filament formation. First, I distinguished between filament formation and ionic accumulation by choosing metal contacts. It was discovered that Silver and Gold produce different memristive JV shapes, with Silver-capped devices exhibiting more pronounced hysteresis.

In the second experiment, I placed a chemically inert layer between Perovskite and metal contact to determine the nature of filament formation. This inhibited any chemical reaction between them, and as expected, the hysteresis was reduced. The JV curve of the Silver-based devices also began to resemble that of the Gold-based devices.

The difference in the JV curves in the first experiment implies a difference in the underlying mechanism of the resistance switching between Silver and Gold-based Perovskite memristors. The results of the second experiment suggest that the filament formation mechanism is activated/enabled via Silver to Perovskite chemical reaction. However, it is unclear what the filament is made of and how it is formed. Therefore, the model needs to be general enough to accommodate any possible underlying microscopic phenomena.

The model was based on very simple principles and geometry. In summary, the filament

is proposed to be a cylindrical structure of a conductive material that can grow inside the bulk Perovskite. The device's conductance is calculated as a parallel connection of two independent regions of the device, represented as resistors, with JV behaviour calculated utilizing Ohm's law. The model relying on simple physics implies that it cannot be used for device physics/material science, as it is not accurate enough for this purpose. However, this simplicity allows for model simulation to run fast, enabling circuit simulation. Another thing of note, the geometric constraints of the filament and the asymmetry of its growth manifest the hybrid nature of the model, making model simulation not straightforward.

To check that the model can indeed replicate the overall behaviour of the real-world device, it needed to be verified against real-world data. To verify the model, I implemented it in the Python programming language and fitted it into the experimental data via parameter optimization. Since the model was able to reproduce the overall behaviour of real-world devices, it was deemed applicable for circuit simulation. As such, it was used in the next stage of this project - Fuzlearn hardware acceleration circuit design.

Chapter 5

Fuzlearn Hardware Circuit

5.1 Introduction

To be applicable for in-situ ML, Fuzlearn needs to be implemented into hardware. While there are already platforms for realising and hosting ML models, e.g. the aforementioned

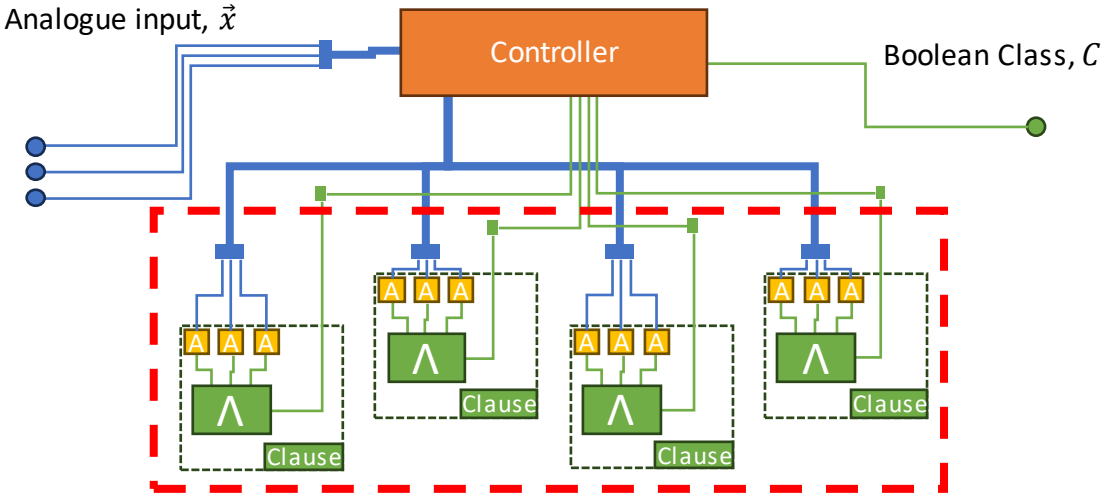


Figure 5.1: Data flow diagram of Fuzlearn hardware implementation. Fuzlearn machine receives analogue inputs from the outside world, processes them, and then returns Boolean class (i.e. a reference Voltage) back to the outside world. The machine is split into two major components: Fuzlearn clause ensemble (enclosed in the red-dotted rectangle) and the controller implementing the operating protocol. Solid coloured lines represent the flow of the data. Blue lines - analogue data. Green lines - Boolean data.

multiPULPly [66], there are still some concerns to be addressed. Namely, the issue is that most of the conventional computing hardware, including PULP [67], follows von Neumann architecture principles [19]. One of the main principles is the separation of data storage and data processing, which introduces latency and additional energy costs due to the data movement. As such, for Fuzlearn realisation I decided to explore an alternative paradigm of computing architecture, called 'in-memory' computing [25].

Figure 5.1 presents a schematic idea of Fuzlearn hardware implementation. Similarly to its mathematical definition, Fuzlearn hardware is separated into two logical constructs: clause ensemble (i.e. Fuzlearn framework) and the controller. In this chapter, I focus primarily on describing the Fuzlearn clause ensemble expressing it as electrical circuits, as it is the primary component that holds the essence of the Fuzlearn machine. A controller, while still being necessary, is treated as a secondary component of the machine, and, as such, will not be discussed in this work. But, as a suggestion for future research, the controller could potentially be realised using the aforementioned PULP platform.

As for the clause ensemble, the presented circuits do not incorporate any specific real-world electrical components, except for the Perovskite memristor for which I made a specific model, see Chapter 4. However, for each theoretical or ideal peripheral electrical component, there could be found a real-world counterpart. By using abstract peripheral components, I aim to prove the general idea of the Fuzlearn hardware - that it can be implemented.

For the purposes of circuit design and simulation, I chose to utilize the LTspice program [142]. LTspice belongs to the SPICE family of circuit simulation software, an acronym for "Simulation Program with Integrated Circuit Emphasis" [143]. What sets LTspice apart from other -spice variants is that it functions as a graphic-based circuit simulator. This allows users to visually draw their circuits instead of defining them in text only, making LTspice circuit design more intuitive and evident, even to those without an advanced electrical engineering background. Additionally, LTspice is freeware and easy to use, which makes it accessible to anyone interested in circuit design.

5.2 Fuzlearn circuit

To transfer the Fuzlearn's mechanism into hardware, its abstract hierarchy outlined in Chapter 3 was used as a blueprint. This involved creating circuit equivalents for each level of Fuzlearn abstraction, starting from the lowest level with a single state variable, implemented using the memristor, followed by Automaton, Clause, and finally finishing with a multi-clause ensemble. For each level of abstraction, custom circuit symbols were created for encapsulation, making it easier to use in later stages. Each stage's corresponding circuits are presented alongside these custom circuit symbols.

5.2.1 LTspice legend

In this Chapter, LTspice circuit diagrams are extensively employed as visual aids. These diagrams predominantly feature encapsulated subcircuits, resembling labelled boxes, that simplify the representation of complex subcircuit elements. However, there comes a point where the most basic subcircuits need to be depicted using fundamental components. Figure 5.2 provides a reference guide which describes the commonly utilized electrical components within LTspice.

5.2.2 Memristor circuit

To include my Perovskite memristor model in circuit simulations, I first had to translate its governing equations into LTspice's framework. I chose to portray the model's intrinsic regions with native electrical components provided by LTspice. Specifically, the entire model is represented as a parallel connection of two resistors that represent the static and variable resistances in the model. The resulting circuit is shown in Figure 5.3(a). To implement variable resistance, I used the arbitrary behaviour element, allowing non-constant behaviour dictated by a user-defined function [144]. To address the hybrid nature of the model and prevent numerical instability issues, I utilized LTspice's enhanced accuracy settings. These settings are detailed in Chapter 6.

After implementing my Perovskite memristor model in LTspice and encapsulating it in

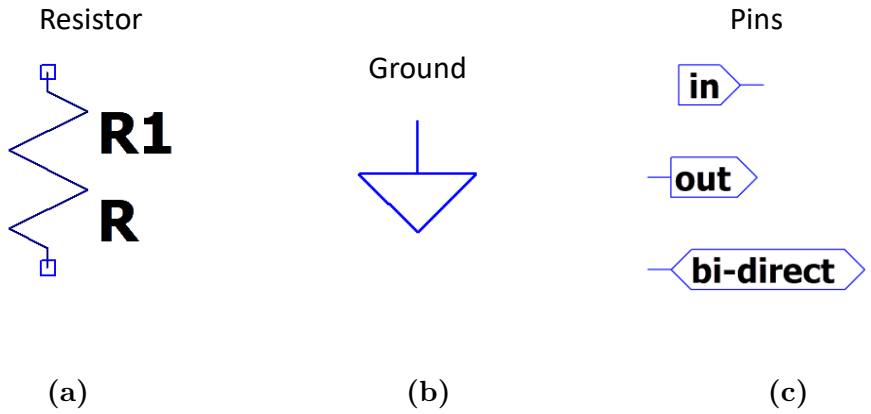


Figure 5.2: LTspice common electrical elements/components. **a)** Resistor. **R1** is the component's label which can be used by the rest of the circuit for reference. **R** is a temporary placeholder for the value of actual resistance. **b)** Electrical ground - a pin/point where the value of electrical potential/voltage is constrained to 0V. **c)** Three different types of pins: input, output, and bi-directional. The label inside the pin can be used by the rest of the circuit to reference the value of the voltage at this pin. The difference between input/output/bi-directional is mostly graphical and is present for convenience only.

a custom symbol, it was tested using the original data. The results of the LTspice-based simulation were found to be almost identical to those obtained from Python computational simulations, see Figure 5.3(c). There are differences as compared to the model's prediction presented in Figure 4.7(b), which, I believe, arose most likely due to LTspice using different algorithms for solving differential equations compared to Scipy. Nevertheless, these differences are minor compared to the overall behaviour, which was successfully transferred from Python model implementation to LTspice. Thus, this LTspice model can be utilized for the design of Perovskite memristor-based devices and integrated circuits.

Looking ahead to future applications, I decided to encapsulate the memristor within two pairs of electrical switches, which are described in more detail in Chapter 6. The circuit of the encapsulated memristor is presented in Figure 5.4. The primary reasoning behind this design choice is to separate the reading and writing functionalities of the memristor, i.e. logically separating the lines for different operations which would be executed upon the memristor. By utilizing separate circuit pathways for the reading and writing operations, I can achieve direct control over the internal resistive state of the memristor, eliminating any

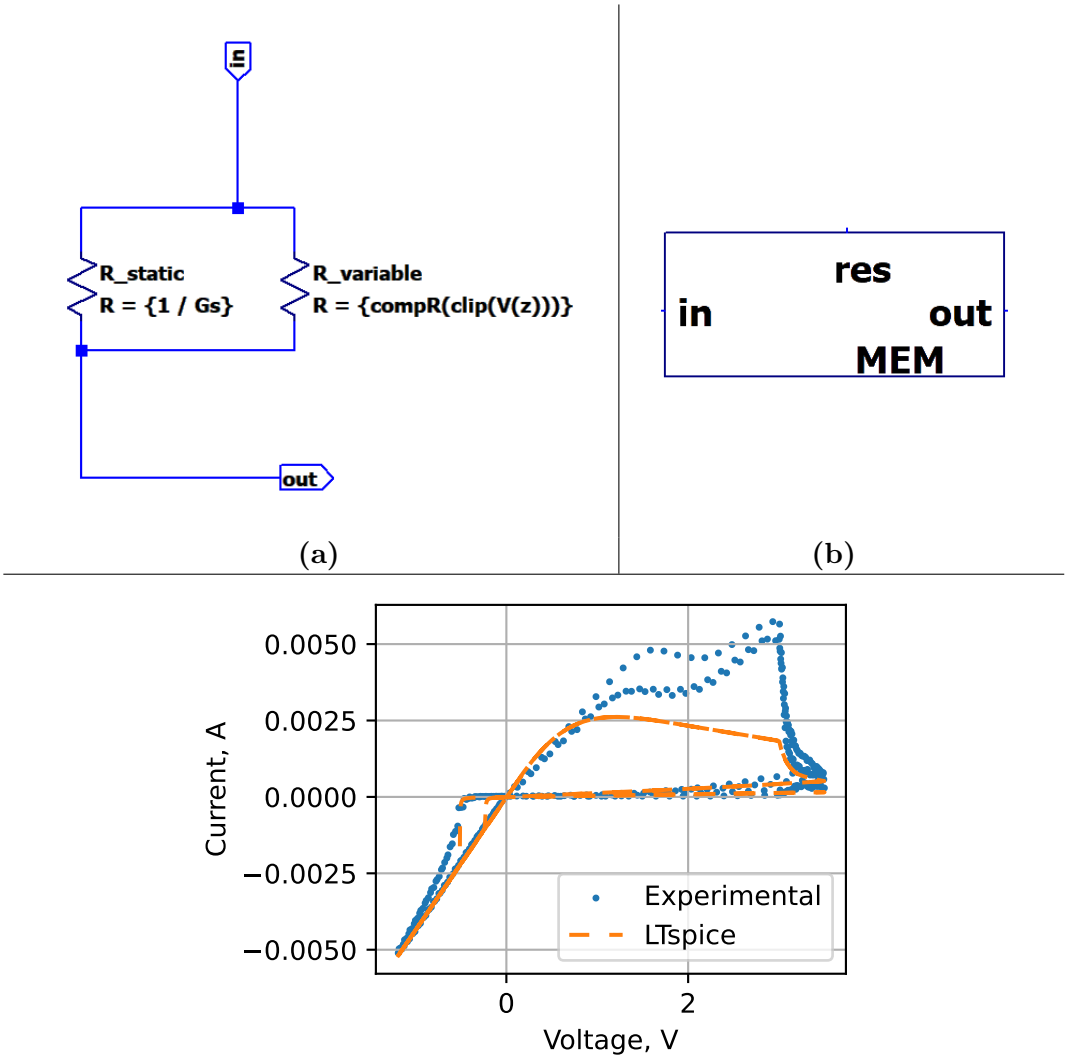
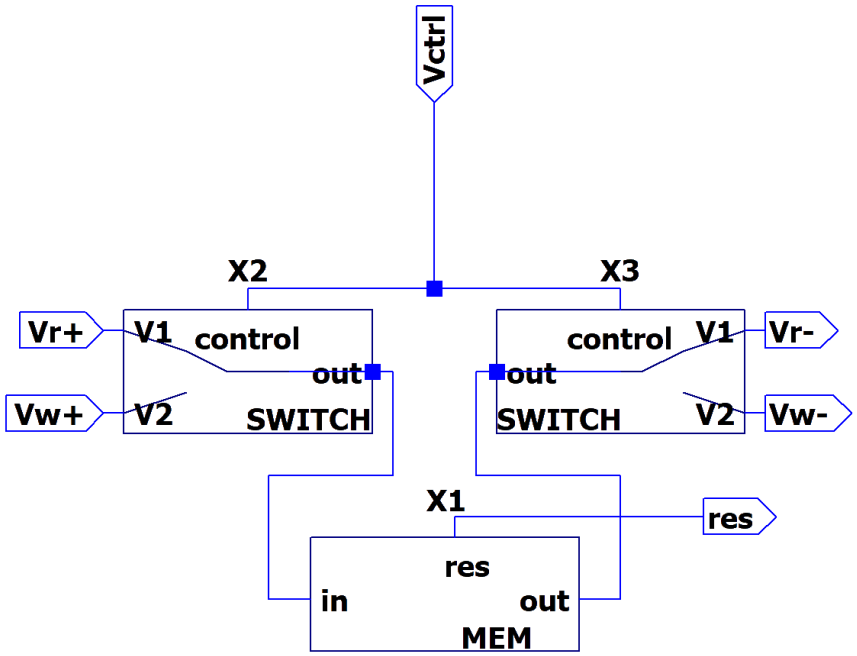
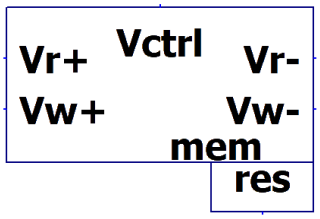


Figure 5.3: Circuit schematic of Perovskite filament formation memristor model implemented in LTspice. **a)** Circuit schematic. Memristor is represented as a parallel connection of two standard resistors. Each resistor utilises the 'expression' feature to calculate the value of their resistances. Pins **in** and **out** are the pins through which the memristor connects to the rest of the circuit. **in** - positive pin, **out** - negative pin. **b)** Corresponding subcircuit symbol for memristor. **c)** LTspice Simulated JV curve of the memristor model vs. original experimental data.

possible concerns related to other elements in the circuit. This encapsulated memristor will serve as a fundamental building block for the Automaton circuit.



(a)



(b)

Figure 5.4: LTspice schematic of Halide Perovskite memristor placed between 2 two-way switches to separate READ and WRITE lines. **a)** Circuit schematic. **X1** - memristor, **X2** and **X3** - two-way switches. **Vctrl** is the controlling pin which dictates whether the memristor's pins are connected to the READ line (**Vr+**, **Vr-**), or to the WRITE line (**Vw+**, **Vw-**). The sign on the label of a line pin represents the polarity of the pin. **b)** Corresponding symbol, which encapsulates this circuit.

5.2.3 Automaton circuit

The next level in Fuzlearn's hierarchy is the Automaton. Similarly to its mathematical counterpart, the main purpose of this circuit is to transform an analogue signal within a specific range into a digital or Boolean signal that can be further utilized by other circuit components.

The general idea for the Automaton circuit is a simple potential divider controlling a switch. In short, a potential divider, or voltage divider, is a simple electrical circuit

that allows one to 'divide' the applied voltage by a constant ratio, which is determined by the resistance values of the constituent resistors [145]. The schematic representation of a standard potential divider can be seen in Figure 5.5(a). Here, the resulting divided

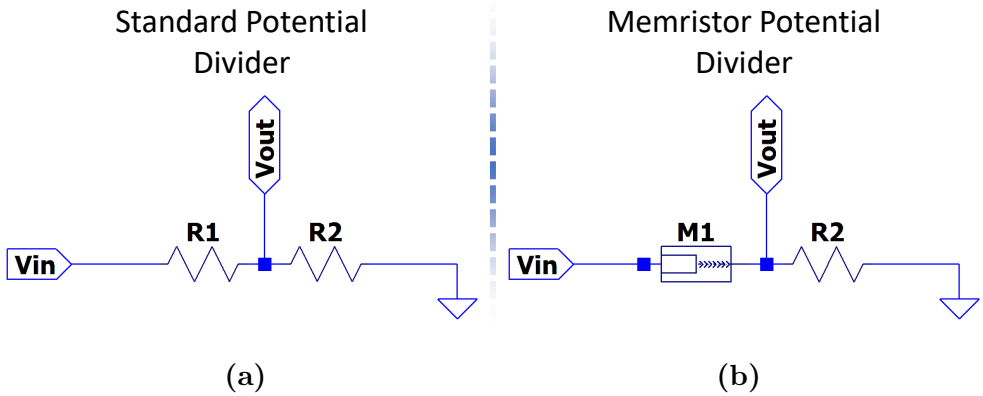


Figure 5.5: Circuit schematic of a potential divider. **a)** Standard, i.e. constant ratio, potential divider. The voltage at pin **Vout** is a function of voltage at **Vin** and the resistances of **R1** and **R2**. **b)** A potential divider with one resistor substituted for a memristor, which allows for a resistance variable ratio.

voltage, denoted as V_{out} is a function of the applied input voltage, denoted as V_{in}. In this case, the output voltage is a constant fraction of the input voltage, where the fraction is pre-determined by the resistance values, according to the formula presented below.

$$V_{\text{in}} = V_{\text{out}} \frac{R_2}{R_1 + R_2} \tag{5.1}$$

However, what if one of the resistors were to be substituted by an electrical component which can vary its resistance? In this case, the output voltage would not only be the function of the applied input voltage, but also the function of the resistance of that said element. This is where the memristor comes into play. The memristor is connected in series with a known constant resistor, creating a variable potential divider, as can be seen in Figure 5.5(b). By varying the resistive state of the memristor, we can achieve different non-volatile division ratios of the applied input bias.

By connecting the divided voltage, V_{out}, to the controlling pin of a switch-like device, e.g. to the gate of a MOSFET or to the base of a bipolar transistor, the switch can act as

a non-volatile memory-driven logical gate for some known voltage level, which represents digital 1 or Boolean `True`. The resulting output of such a digital gate can be expressed as

$$O = V_{\text{ref}} \wedge S\left(V_{\text{in}}, M_1\right), \tag{5.2}$$

where V_{ref} is the reference digital voltage that represents `True`, and S is the state of the switch, either open or closed. The state of the gate is, therefore, a function of the voltage applied to the divider, V_{in}, and the resistance of the memristor M_1. Translating it into the machine learning terms, the level of V_{in} can represent the input data, while M_1 is the internal state of a learning machine. In essence, this is 'in-memory computing' - the same device stores the memory and performs computation based on that memory, with no need for data transfer [25].

By carefully adjusting the switch settings and the divider properties, it is possible to create a simple comparator element with a variable limit, which outputs `True` if the input is lower than said limit. Combining two of these comparators together in a series allows us to replicate the behaviour of the Fuzlearn Automaton. Figure 5.6 presents the final circuit for the Fuzlearn Automaton implemented in LTspice. Here, you can observe the configuration consisting of two switches connected in series to a known reference voltage. One of the switches operates in enhancement mode, while the other functions in depletion mode, for more details, see Chapter 6. When both switches are open, the output voltage matches or is insignificantly lower than the reference voltage. Conversely, if at least one switch is closed, the output voltage becomes nearly zero.

Each switch is controlled by a voltage divider comprising an encapsulated memristor and a reference resistor. The memristor's READ contacts are connected in parallel to the analogue input, V_{read}. When an input signal of the appropriate level is applied, both switches remain open ensuring that the voltage levels at the reference and the result pins are the same. The range of input levels that maintain the switches in the open state is determined by the state of the memristors. In other words, the resistances of the memristors regulate the boundaries for converting the analogue signal into a Boolean

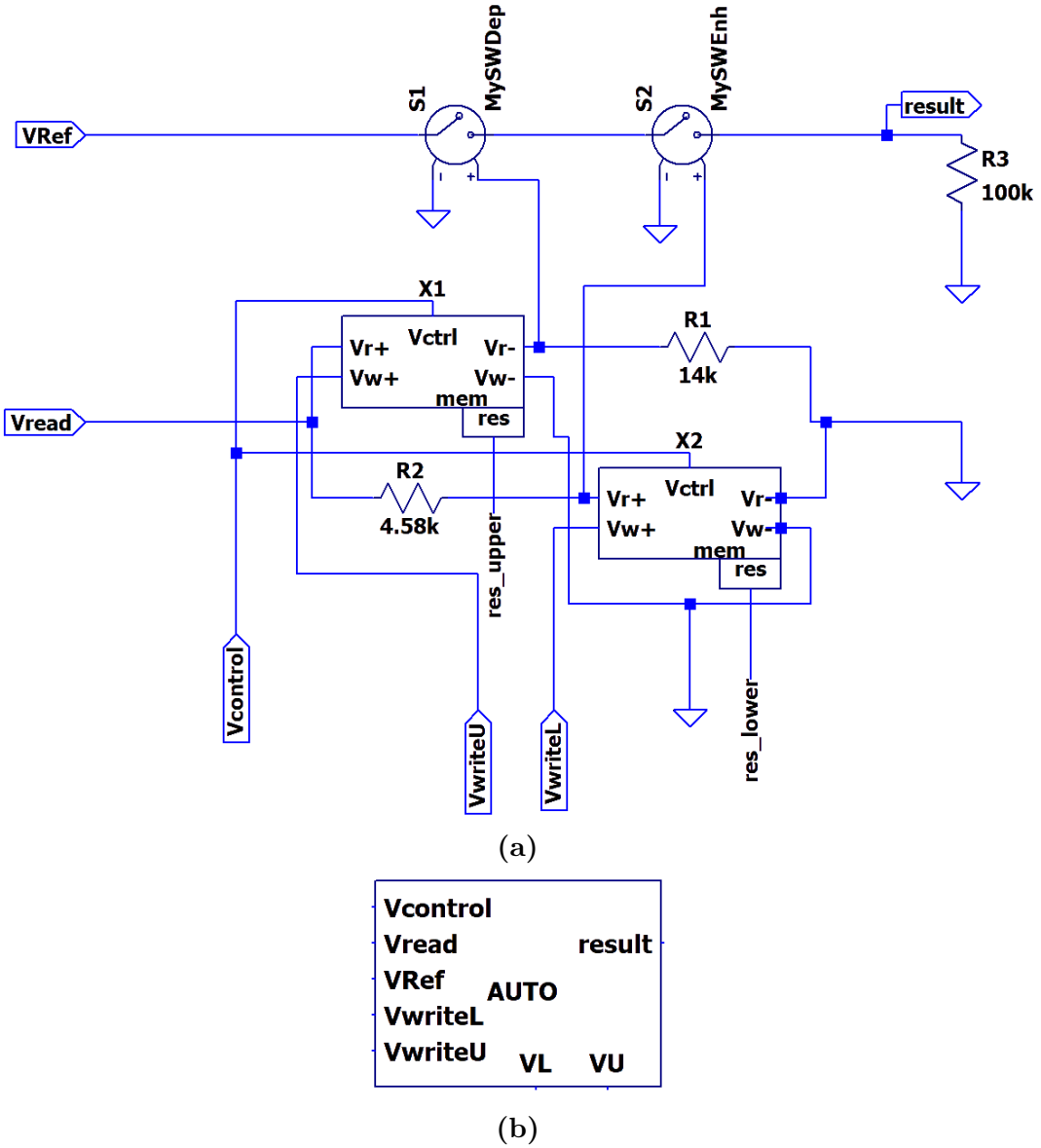
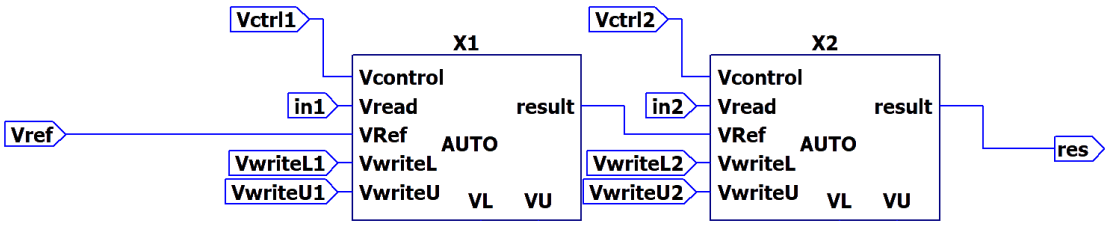
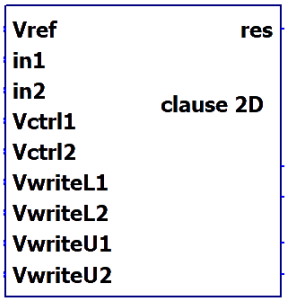


Figure 5.6: LTspice schematic of Fuzlearn Automaton **a)** Circuit schematic. The voltage at the output pin **result** is determined by the voltage level at the reference at **VRef** and the state of the switches, **S1** and **S2**. The state of **S1**, i.e. open or closed, is controlled by the potential divider comprised of **X1** and **R1**. Similarly, **S2**'s state is controlled by **X2** and **R2**. Memristor **X1** and resistor **R1** create a potential divider, whose divided voltage controls a depletion-type type switch **S1**. The voltage at **Vcontrol** dictates whether the Automaton is performing inference, or is being written to. **b)** Corresponding symbol, which encapsulates this circuit. **VL** and **VU** - pins, whose voltages represent the corresponding values of Automaton's L and U for debugging purposes.

representation/reference.



(a)



(b)

Figure 5.7: LTspice schematic of a $N = 2$-dimensional Fuzlearn Clause.**a)** Circuit schematic. The conjunctive product of two Automata, **X1** and **X2**, is achieved by feeding voltage at **result** pin of **X1** to the **VRef** pin of **X2**. As such, the final output at pin **res** equals **Vref** only if switches in both Automata are open, i.e. classify `True`. The input is encoded as voltages at pin **in1** and **in2**. **b)** Clause subcircuit symbol.

5.2.4 Clause circuit

To classify more than one analogue input into a Boolean class, several Automata have to be combined into a Clause. Here, I present a two-input Fuzlearn Clause, i.e. $N = 2$, refer to Figure 5.7. In its original definition, a Clause is a conjunctive product of the individual results of its constituting Automata. To implement such a product, I connected Automata in series, forming a chain-like structure. In this configuration, the reference voltage of one Automaton is provided by the output of the previous Automaton. If any of the Automata within the series classifies its respective input as `False`, the overall output of the Clause will also be `False`. Conversely, if all Automata outputs are `True`, then the final result is also `True`.

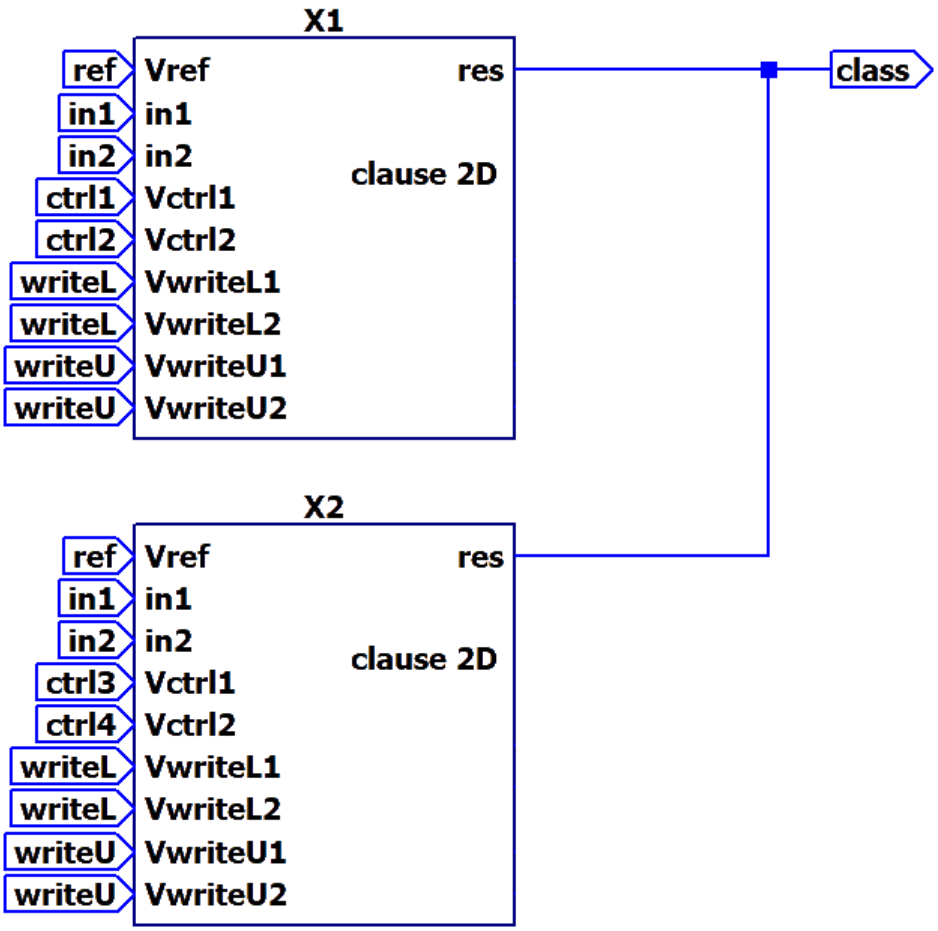


Figure 5.8: LTspice schematic of a two-dimensional two-clause Fuzlearn ensemble. The final result, represented as the voltage at pin **class**, is `True`, i.e. equal to the voltage at **ref**, when either clause **X1** or **X2** classify the point, encoded as voltages at pins (**in1**, **in2**), as `True`.

5.2.5 Machine circuit

To allow for a finer detail in the classification scheme, more than one Clause has to be combined into a Machine. Here, I present a two-clause two-input Fuzlearn machine. The Machine circuit is constructed by combining multiple Clauses, connecting their outputs to a common point, see pin `class` in Figure 5.8. This circuit provides a clear understanding of the working mechanism of the entire Clause ensemble. If any of the Clauses within the Machine produces a `True` output, that specific `True` output becomes the overall output of the entire machine at that particular point. Conversely, only when all Clauses output `False`, the value at the designated point will also be `False`. This behaviour is akin to

logical disjunction, also known as **OR** operation.

This configuration of interlinked Clauses within the Machine circuit enables collective decision-making. The outputs from each Clause are combined to determine the overall outcome of the machine, making it capable of solving complex problems by evaluating the input against multiple conditions simultaneously. The Machine circuit, as depicted in Figure 5.8, will later be employed in tests against a simple XOR problem, verifying its viability as Fuzlearn hardware implementation and showcasing its application in solving logical challenges.

This circuit is the backbone of the Fuzlearn architecture in hardware. However, as with software implementation, it is only a framework without any driving force. To make it work, a scheme of regulating signals needs to be applied to the controlling pins of the circuit, essentially implementing an operating protocol. In the real world, this could be accomplished by an external controlling device, but designing such a thing is outside the scope of this project. Therefore, for the purposes of testing the circuit, an LTspice voltage source with a pre-defined signal scheme will be used to implement the operating protocol.

5.3 Machine circuit testing

With the Fuzlearn machine circuit in place, it was necessary to verify that it is a viable implementation of Fuzlearn in accordance with its theory outlined in Chapter 3. To do so, I conducted tests on the two-input two-Clause machine circuit, presented in the previous section, against simple XOR problems. As such, this section outlines the results of this testing. In the first test, my goal was to assess the circuit's ability to hold Fuzlearn's state and utilize it for inference. For the second test, I focused on the circuit's behaviour in learning conditions, i.e. with its internal state varying during the process.

5.3.1 Loading and Inference test

In this specific test scenario, the boundaries of the Clauses in the Machine circuit were adjusted using predetermined values. Following the adjustment, a set of input signals was

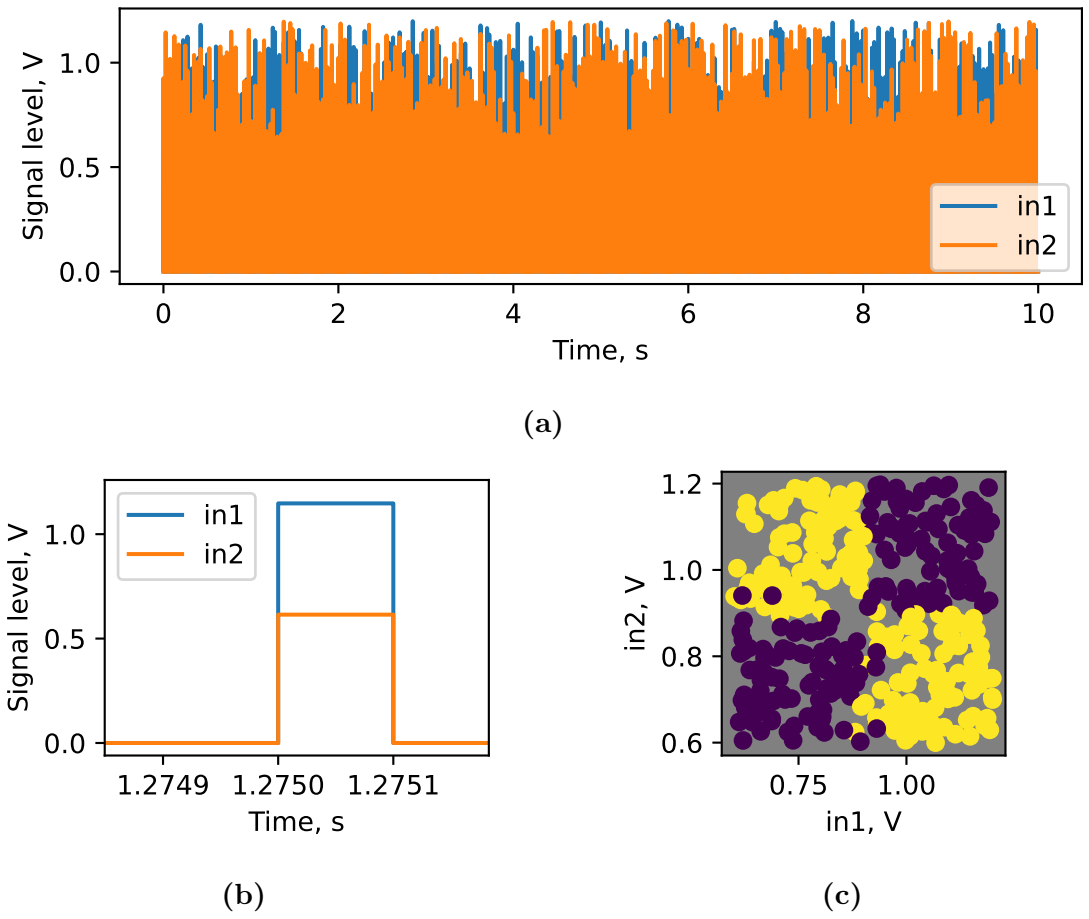


Figure 5.9: Fuzlearn machine circuit testing against XOR - the case of inference. **a)** Input waveforms for the entire experiment. Values of the peaks were generated using Numpy's random number generator. **b)** Zoomed in on a single input signal, the value range for both $in1$ and $in2$ is within $[0.6, 1.2]$ V. **c)** Machine-produced class scattered according to its place in the input space. Yellow - `True`, Purple - `False`.

applied, and the resulting Machine outputs were recorded. A scatter plot was generated to visualize and analyse the obtained data, as shown in Figure 5.9(c). The plot provides a clear representation of the Machine's classification performance, highlighting that the majority of inputs were correctly classified.

However, it is worth noting that a few points were misclassified. Interestingly, the misclassified points share a common characteristic – their proximity to the boundaries of the Clauses. Based on this observation, a hypothesis emerges that even the READ signals in the circuit have the potential to slightly modify the resistive states of the memristors.

This alteration leads to a shift in the boundaries, consequently resulting in misclassification. This finding is significant as it sheds light on a potential challenge that could be encountered in real-world applications. To address this issue, one potential solution could involve readjusting the input parameters of the READ signals, such as the time-width of the signals. However, it is essential to consider that such a problem should be tackled within the context of the actual application, accounting for all circuit elements, substituting hypothetical switches with real ones, and so forth. Solving this problem solely during the hypothetical "proof of concept" stage might not be worthwhile, as it may overlook crucial factors present in a practical scenario.

Taking into account the previous discussions and considerations, overall, I view the results of this test as a success, despite the encountered issues. In terms of replicating the intended theoretical Fuzlearn behaviour, I believe that the circuit accomplished this task with the overall accuracy being $\approx 96.7\%$.

5.3.2 Learning test

Here, I aimed to test how Fuzlearn's hypothetical hardware would behave during an active learning procedure. As such, a randomized training set was generated and provided as input to the Fuzlearn clause ensemble circuit, presented in Figure 5.8. The signals and voltages at the controlling pins, **Vctrl1** and **Vctrl2**, were adjusted manually, thus, simulating the operation of the controller unit. As such, the role of the operating protocol was filled by a predetermined signal set, which was calculated in accordance with the training set. During the procedure, after each misclassified input point, the circuit received an adjusting write signal to its Automata. This adjusting signal ensured that the circuit was trained to correctly classify the currently presented point. Consequently, this simulation emulated the learning procedure within the circuit.

The outcomes of this learning procedure are displayed in Figure 5.10. The figure illustrates how the boundaries of the circuit and, subsequently, the classification of the input space varied over time. Gradually, the boundaries converged to represent the XOR pattern, resembling a two-by-two chequered pattern.

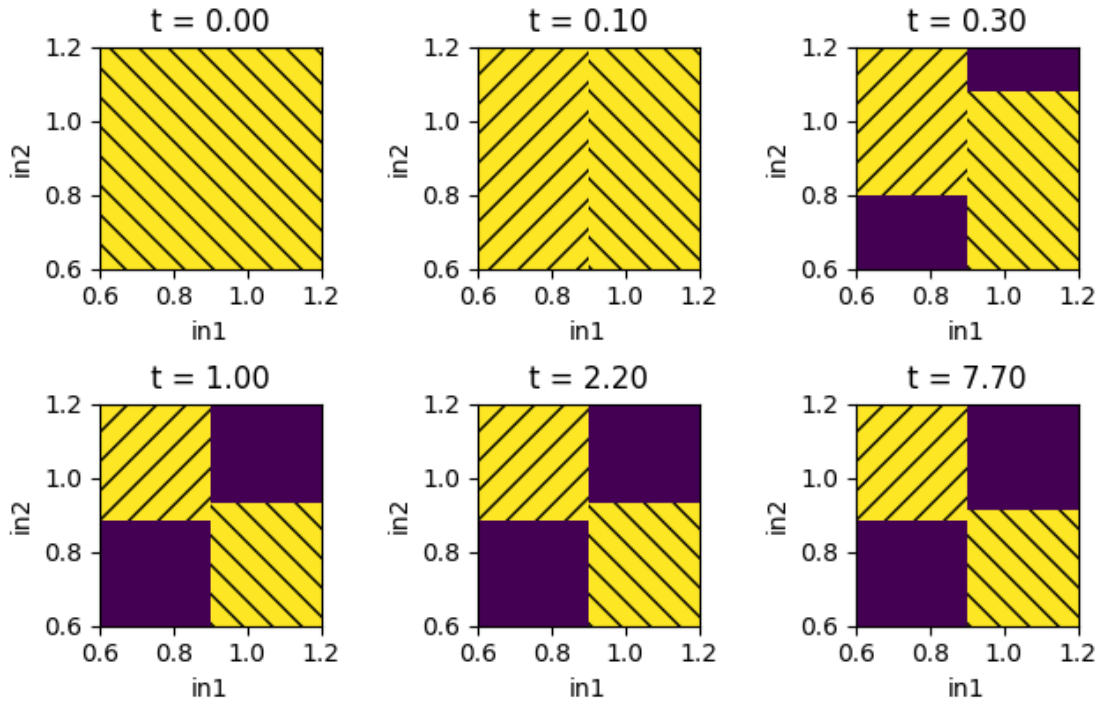


Figure 5.10: Fuzlearn machine circuit testing against XOR - learning process. Each image represents a snapshot of two clauses' boundaries at particular timestamps of the learning procedure. The yellow-hatched area represents the boundaries of each clause, solid purple - `False` area with no clause coverage.

These results demonstrate that the presented circuit serves as a viable hypothetical hardware implementation of the Fuzlearn architecture. The successful convergence of the boundaries and the accurate classification of the XOR problem confirm the circuit's ability to replicate the desired behaviour outlined by the Fuzlearn theoretical framework effectively.

5.4 Conclusion

In this chapter, a hypothetical circuit design was presented for the implementation of Fuzlearn hardware within the LTspice circuit simulation software. The general idea was to prove the viability of Fuzlearn in hardware by converting the mathematical entities of Fuzlearn into their respective circuit equivalents. The incorporation of the Halide Perovskite memristor into the Learning Automaton design was a key feature that made

the Automaton both the memory unit and the processing unit, effectively realizing in-memory computing.

The circuit design was finalised with Fuzlearn machine circuit tailored for two-input analogue-to-Boolean classification tasks, which tested for inference and learning within the LTspice framework. The results of both tests confirmed that the circuit conforms to Fuzlearn's theoretical model. However, it should be noted that the presented circuits cannot be directly turned into real-world hardware due to their use of generalized or ideal peripheral components.

In conclusion, the presented work on circuit design for Fuzlearn's hardware implementation can serve as a starting point for future developments in the field of in-situ machine learning. It can also be used as a template for an actual hardware integrated circuit (IC) or printed circuit board (PCB).

Chapter 6

Methodologies

This chapter differs from the previous ones as it focuses on the methods and procedures used in the project, rather than just presenting the project itself. The purpose of this chapter is to provide valuable guidance for any future continuation of the project. It serves as a foundational stepping stone, introducing important key aspects that were instrumental throughout the project. These aspects are detailed below.

As I was heavily involved in mathematical modelling and numerical simulations for this project, I would like to provide more information about the programming tools I utilized. This includes the software I used for both Fuzlearn's modelling in Chapter 3 and Perovskite memristor model simulations in Chapter 4. This involves a concise rundown of the programming language I opted for, namely Python, along with the libraries associated with it, that helped me with my achieving my work. Based on the detailed explanation of Python and its libraries, I aim to provide a more comprehensive understanding of the implementation of Fuzlearn's software. This task was not easy and involved numerous versions and attempts to achieve the perfect outcome. In this regard, I have included the code snippets that demonstrate the critical components of the final and most optimal version.

In addition, this chapter outlines the experimental and fabrication methods used in my work on Halide Perovskite memristors, as seen in Chapter 4. It covers all the necessary steps to create a Perovskite memristor, along with a thorough explanation of the custom-

made characterisation kit used to measure my devices, and, finally, it details the specifics of the characterisation procedure itself.

A dedicated section also addresses the usage of LTspice, encompassing its application, auxiliary circuits incorporated within the Fuzlearn circuit design, and a concise explication of the key components. This serves to facilitate the reader's comprehension of the circuit design process within the LTspice environment.

In conclusion, the chapter discusses the use of various instruments that were utilized throughout the project. Specifically, language-assisting tools were employed to aid in scientific writing due to English not being my first language.

6.1 Python programming

During this project, I utilised Python extensively [92]. It is an interpreted, dynamically typed programming language that is user-friendly and requires a minimal amount of code to achieve significant outcomes. Despite being a high-level language, Python is adaptable and can act as a wrapper language for existing efficient machine code, which is usually realised as high-performance computing libraries. This allows for smoother execution with minimal computation delays. For the numerical modelling and simulations, I used various libraries that take advantage of this capability, which enabled me to use Python's concise and high-level syntax while benefiting from the efficient low-level implementations underneath. This combination of ease of use and computational performance helped me achieve a harmonious balance in my work.

6.1.1 Numpy

Numpy [146] is the Python code library with a wide range of applications that enables rapid computation through parallelized array arithmetic [147]. Its main benefit is in simplifying basic operations on matrices, vectors, and arrays, making it ideal for linear algebra tasks, Monte Carlo simulations, and other numerical computational methods. Numpy is highly efficient and achieves this by using pre-compiled FORTRAN code [148]. This blend of

efficiency and functionality makes it a valuable asset in my toolkit.

Numpy's functionality is similar to MATLAB, another programming language for science [149]. However, unlike MATLAB, the underlying Python provides a more solid groundwork for managing programming projects and connecting scientific computing algorithms with a broader range of general-purpose programming tasks, which is particularly advantageous in situations where MATLAB can become cumbersome [150].

6.1.2 Scipy

Scipy [151] is a complementary 'sister' library to Numpy. While Numpy forms the foundation of numerical operations, Scipy takes a step further by offering an extensive range of advanced algorithms that cater to diverse scientific and computational needs [129].

Scipy's scope encompasses a multitude of intricate scientific algorithms that surpass in complexity the numerical methods present in Numpy. These algorithms not only address fundamental numerical operations but also venture into complex realms. Some of these sophisticated algorithms involve the field of numerical integration, enabling accurate computation of integrals over various ranges [152]. Additionally, Scipy includes several interpolation frameworks ranging from simple linear 1D interpolation and finishing with multi-dimensional Splines [153]. These algorithms facilitate the creation of continuous functions from discrete data points, aiding in the smooth approximation of data between known values.

Moreover, Scipy boasts a robust collection of Ordinary Differential Equation (ODE) solvers [131], including the well-known Runge-Kutta method, which was previously mentioned in the context of the model parameter fitting in Chapter 4. These solvers are indispensable tools for modelling dynamic systems and simulating their behaviour over time, contributing to a wide array of scientific and engineering disciplines.

Beyond its prowess in numerical analysis, Scipy presents a treasure trove of optimization algorithms, encompassing both local and global optimization strategies. Notably, the dual annealing method [134], highlighted in the same memristor chapter, was an indispensable tool for this project.

In essence, Scipy is an essential tool for numerical computation, offering advanced algorithms for scientific challenges. Its integration, interpolation, ODE solvers, and optimization strategies empower researchers across domains.

6.1.3 Matplotlib

Matplotlib stands as another essential Python library [154]. Its primary focus lies in creating graphs and plots, making it an invaluable tool for visualizing data and conveying insights in a straightforward manner. Matplotlib has a noteworthy characteristic of integrating smoothly with both Numpy and Scipy. This seamless interaction permits users to effortlessly convert numerical data into informative visual representations. The integration between these tools bridges the gap between numerical computation and visual communication, enabling a comprehensive data analysis and presentation approach.

In this thesis, I utilized Matplotlib to create various graphical illustrations such as curves, scatter plots, and heatmaps. Specifically, the `pyplot` subpackage played a crucial role in this task [155]. It not only simplified the graphing process but also made it more accessible and intuitive by introducing a higher level of abstraction.

6.2 Fuzlearn software implementation

In Chapter 3, a comprehensive description of Fuzlearn is provided that outlines its mathematical and logical underpinnings, as well as its operational principle. While that chapter thoroughly delves into the conceptual framework, there exist noteworthy aspects concerning the software implementation of Fuzlearn. This facet is explored in greater depth in this section, providing readers with a comprehensive insight into this crucial aspect. In this section, in particular, the core components of the latest version of Fuzlearn's software implementation in Python are presented.

For this software implementation, I employed an Object-Oriented programming approach [156], which neatly encapsulated the core behaviour of the entire Fuzlearn machine into a single organized and manageable unit, see the first code snippet as depicted in Fig-

ure 6.1. Namely, the entire Fuzlearn machine is implemented as a single class, named `LGM`. From a programming perspective, the entirety of the Fuzlearn machine's characteristics can be defined with a few variables: the number of input dimensions - `size`, the number of subdivisions per dimension - `subs`, and, finally, by the array of clauses that the machine employs. Together, these logical components fully define the machine's behaviour within its operational context. As shown by the very first line of this snippet, the machine's implementation relies on Numpy.

```
1  import numpy as np
2
3  class LGM:
4
5      def __init__(self, size, subs=2):
6          self.size = size
7          self.subs = subs
8          self.clauses = {}
```

Figure 6.1: Fuzlearn core code snippet. Fuzlearn machine is implemented as a Python class, named `LGM`. Class method `__init__` implements object initialisation, which creates an instance of the class and fills it with attributes. The entire machine is defined by the number of input dimensions, `size`, the number of subdivisions per dimension `subs`, and the dictionary (map) of all employed clauses.

Since the number of clauses for the machine can vary greatly based on the number of dimensions and subdivisions, to not waste memory space, I used a dynamic key-value container, commonly referred to as a *dictionary* [157], which only allocates memory for new clauses when necessary. The clause instantiation code can be seen in Figure 6.2. As seen

```
1      def compcell(self, arr):
2          return np.where(arr < 1.0, np.floor(arr * self.subs), self.subs - 1.0)
3
4      def spawnclause(self, key, coords):
5          clause = np.empty((2, self.size))
6          clause[0, :] = coords / self.subs
7          clause[1, :] = (coords + 1.0) / self.subs
8          self.clauses[key] = clause
```

Figure 6.2: Fuzlearn core code snippet. Function `compcell` calculates the grid cell coordinates to which a particular input vector, `arr`, belongs. Function `spawncell` performs clause allocation. A clause is a $2 \times N$ array that is stored in the `clauses` dictionary.

in this code, a Clause is represented as a $2 \times N$ array of floating-point numbers, where N is the number of dimensions. Each value in this array is filled with the corresponding value of the Automaton's boundary limit, either lower or upper depending on the first index. This

representation of a clause is particularly beneficial since it avoids the creation of additional abstractions, which limits the total amount of code and helps with overall readability. In addition, since a clause is represented as a `numpy` array, all actions involving clauses can be implemented in terms of array arithmetic, which ensures the code's efficiency.

6.2.1 Multiprocessing

When dealing with long training sessions with datasets like Ozone and HTRU2, an optimization approach is needed to speed up the process. To make this happen, multiprocessing was used. To achieve it, I used the built-in Python library called `multiprocessing` [158]. This is an important tool that allows the same Python script to run concurrently across multiple processors or cores. By using this capability, computational tasks can be distributed and processed in parallel, saving a lot of time.

```
1  if (__name__ == '__main__'):
2      from random import getrandbits
3      from multiprocessing import Pool
4      M = 500
5      seeds = [getrandbits(32) for i in range(M)]
6      start = time()
7      with Pool(4) as pool:
8          res = np.array(pool.map(do_round, seeds))
9      end = time()
10     d = end - start
11     print(f'{d} : {d / M} : {d / M * 500}')
12     np.save('compdata\\htru2\\htru2', res)
```

Figure 6.3: Fuzlearn training code snippet with multiprocessing. Object `pool` is a pool of 4 workers each capable of running functions concurrently and independently. Function `pool.map` instructs the workers to run function `do_round` for each and every element of `seeds` list, a list of random seeds. Function `do_round` performs a single training round/session.

The `multiprocessing` library boasts a comprehensive framework that caters to various levels of multiprocessing complexity and abstraction. To keep the underlying multiprocessing code simple to comprehend, I decided to offload the resource management and allocation to the Python environment itself. This choice led to the adoption of the `Pool` abstraction, which streamlines the process and focuses on the core logic of the script [159]. Within the `Pool` abstraction, a virtual 'pool of workers' is created, each capable of executing the designated tasks from a shared stack. These workers function as independent

entities, each empowered to execute a distinct Python script concurrently. This division of labour across multiple cores can facilitate enhanced performance, enabling multiple tasks to progress simultaneously.

The dynamics of the `Pool` operation are notably flexible and adaptive. Depending on the computational demands of the assigned scripts, the `Pool` regulates the allocation of cores to workers. In certain scenarios, each worker might be allocated its dedicated core, ensuring optimal parallelisation. However, there could be instances where workers share cores. Such allocation scheme accounts for potential overhead delays associated with the management of additional cores, which can sometimes outweigh the computational gains achievable on multi-core processing.

In essence, the integration of the `multiprocessing` library, particularly through the `Pool` abstraction, enabled me to accelerate the Fuzlearn machine training process. By effectively harnessing the capabilities of multiple cores, the optimization process ensures the efficient execution of tasks, exemplifying a balance between parallelisation and core allocation. The code snippet, which is presented in Figure 6.3 illustrates the implementation of this multiprocessing strategy.

6.2.2 Github repository

I have set up a dedicated repository on GitHub for Fuzlearn 's Python implementation for a global audience to see and utilise [160]. Contained within this repository are several essential components that collectively contribute to the accessibility and utility of the Fuzlearn implementation. Central to this collection is the core file, namely `src/fuzlearn.py`, which forms the bedrock of the Fuzlearn machine's definition, and which was partially discussed in the previous subsection. This file encapsulates the logic that drives the functionality of Fuzlearn, providing a comprehensive understanding of its inner workings.

Moreover, the repository includes an example script, titled `iris_training.py`. This script serves as an illustrative guide, showcasing how to effectively train the Fuzlearn machine using the Iris dataset. By studying this example, users can glean insights into the practical implementation of Fuzlearn and gain a hands-on understanding of its training

process.

6.3 Perovskite Memristor Fabrication

In this section, I provide the details of the process I used to create Perovskite memristors for my project. The main goal of this section is to provide a comprehensive guide that can help other researchers to follow the path that I have established if they wish to further research in this area.

It is important to note that the techniques used to create Perovskite solar cells are similar to those used in the fabrication of Perovskite memristors. This is not surprising since photovoltaics are currently the primary application for Halide Perovskites. Therefore, the familiarity with these fabrication methods in the field of photovoltaics naturally extends to the emerging field of Perovskite memristors. The next subsections clearly explain the complete process of creating Perovskite memristors in logical order.

6.3.1 MAPI solution preparation

Methylammonium lead iodide (MAPI) was the Perovskite of choice for this project. To prepare MAPI solution for memristors, 750 mg of lead iodide (PbI2) and 260 mg of methylammonium iodide (MAI) are to be dissolved in a 1 mL of 4:1 mixture of dimethylformamide (DMF) and dimethyl sulfoxide (DMSO). The resulting mixture is to be placed on a hot plate set at 100°C and left until complete dissolution of salts, i.e. when the solution becomes clear. Approximately, it takes 15 minutes for the solution to become clear. Once dissolved, the solution is to be filtered using a hydrophobic syringe filter.

6.3.2 PMMA solution preparation

Polymethyl methacrylate, commonly referred to as PMMA, is a polymer that was used in one of the experiments. In order to create the PMMA solution, 500 µg of PMMA are to be combined with 500 µL of chlorobenzene (CB). The mixture is to be heated on a hotplate at 100°C, the same way MAPI was, i.e. until the solution becomes clear.

6.3.3 FTO laser etching

To create my Perovskite memristors, I used the glass slides, pre-coated with fluorine-doped tin oxide, or FTO, as a foundation. FTO is a transparent conductor, which is used extensively in the field of photovoltaics. These slides are commercially available, which, in turn, ensures consistency across batches of devices.

Before the Perovskite layer can be applied onto the substrate, a specific pattern needs to be etched onto the FTO. The pattern separates different devices, commonly referred to as pixels, on a single substrate so they are not electrically connected through the FTO layer. The etching pattern is shown in Figure 6.4. To etch the pattern, I utilised Rofin EasyMark Laser marker model IV F20, which applies periodic laser pulses onto the material. This is a commercially available laser marker.

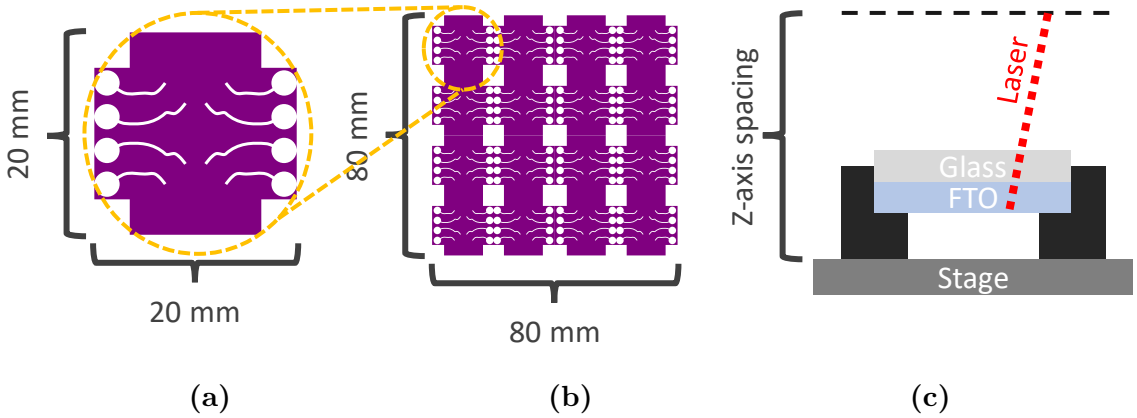


Figure 6.4: Laser etching pattern for FTO substrates. **a)** A single substrate pattern for a substrate size of 20mm x 20mm. Coloured regions are removed, and white regions are left intact. **b)** Full glass slide pattern 80mm x 80mm contains 16 single substrates. **c)** Schematic representation of the etching process. FTO slide is placed 'face down' on a slide holder to allow etched FTO to fall down from the slide.

To etch FTO with the marker, I utilised the following laser parameters:

- 18 mm Z-axis spacing
- 2.2 mm slide holder (distance between the stage and FTO end)
- 50 kHz Pulse frequency

- 500 mm / s laser scanning speed
- 20% pumping power
- 0.01 line width

As shown in Figure 6.4, the FTO etching pattern incorporates a small footprint for the devices, ensuring that the device's bottom contact covers fewer possible defects in the Perovskite layer. Furthermore, the devices are positioned closer to the centre of the substrate, where the film is supposed to exhibit a greater level of uniformity and higher quality.

After the etching process, the substrates can be stored indefinitely since both FTO and glass are highly stable materials. As such, it's common to pre-etch a substantial quantity of these slides to avoid the etching overhead in the fabrication process.

6.3.4 Substrate cleaning

Before the Perovskite layer can be applied onto the etched glass-FTO substrate, it has to undergo a meticulous cleaning procedure. This cleaning process can be divided into two logical stages: the wet stage and the dry stage.

In the wet stage, substrate cleaning is performed within a fumehood, and it involves the utilization of an array of solvents and chemicals. First, a chemical solution known as Hellmanex is employed [161]. Hellmanex is an alkaline solution, which can be prepared from a commercially available concentrate. This solution has been intricately designed to address the cleaning requirements of various glass surfaces, including lenses and flat glass substrates. The method for using Hellmanex involves applying the solution onto the FTO-coated side of the substrate. The application is followed by a gentle scrubbing action, which can be done with a cotton bud or, as discovered within our research group, a more effective alternative — utilising toothbrushes for scrubbing. Once the cleaning with Hellmanex is completed, the substrate undergoes successive rinses. This entails rinsing with Acetone, followed by a rinse with Ethanol, and a final rinse with Deionised water. To expedite the drying process and to prevent the formation of water spots, a nitrogen

gun is employed to rapidly remove any residual liquid from the substrate's surface. With this, the wet stage of the cleaning process comes to its conclusion.

Transitioning to the dry stage, the cleaned substrates are introduced into a UV-O_3 cleaning machine. The particular machine I used in my project is Ossila UV Ozone Cleaner [162]. Each cleaned substrate is housed within this machine for a duration of 15 minutes, ensuring a thorough and effective ozone treatment.

6.3.5 MAPI and PMMA spincoating

Following the cleaning procedure, the substrates are carefully transitioned into the glovebox, which maintains an inert nitrogen atmosphere, effectively shielding the substrates and solutions from any potential reactions with oxygen and moisture. To apply layers onto the substrate, I employed spincoating - a widely used technique, wherein the target solution is dropped onto the spinning substrate [163]. Such a method ensures the homogeneity and thickness of the resulting film. To facilitate the process of spincoating, I employed a Laurell WS-650-23 spincoater, which was placed within the confines of the glovebox.

Regarding the spincoating of Methylammonium Lead Iodide (MAPI), the procedure entails the precise deposition of 50 µL of MAPI solution onto the rotating substrate with a mechanical pipette. The spincoating program is as follows. Beginning with a spin speed of 1000 rpm for a duration of 10 seconds during which MAPI solution is dropped, this initial phase ensures an even distribution of the MAPI solution across the substrate. Subsequently, the spin speed is elevated to 5000 rpm for a duration of 30 seconds. This elevation serves to thin down the solution layer, achieving a targeted film thickness of approximately 150 nm. In a pivotal moment during the spincoating program, 300 µL of Chlorobenzene (CB) is dropped onto the substrate surface. This step occurs 23 seconds before the end of the program. This event kickstarts the crystallization process of the MAPI layer, ensuring its resulting smoothness and glossiness.

After the spincoating, the substrates undergo an annealing process on a hot plate. The initial annealing stage starts at 40°C and spans a duration of 45 minutes, followed by an additional phase at 100°C for 10 minutes. After annealing, the substrates are removed

from the hot plate and left to cool down to room temperature.

For the devices where a PMMA layer was placed in between the top metal and the Perovskite, the following PMMA spincoating procedure was performed. After the MAPI-coated substrates are cooled down, the PMMA layer is spincoated onto the substrates in a similar manner. A total of 300 µL of PMMA/CB solution is deposited, while the substrate is spinning at 5000 rpm for 30 seconds. Post-spincoating, the substrates are once again returned to the hot plate for additional annealing of the PMMA layer. The annealing, again, takes place at 100°C for an additional 15 minutes. With the completion of the annealing process for the PMMA layer, the substrates are allowed to cool back to room temperature, rendering them primed and prepared for the subsequent phase of metal contact evaporation. Thus, the main part of the fabrication process is concluded.

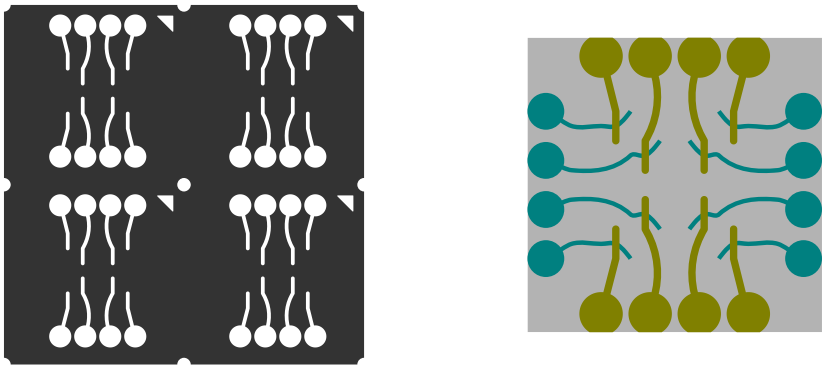
6.3.6 Metal contact thermal evaporation

Following the spin-coating of the layers, the samples were introduced into the thermal evaporator to deposit metal contacts. I employed the MBraun in-glovebox thermal evaporator system to execute the evaporation of metal contacts. The outcome was a thin layer of the intended metal on the substrate, with a thickness in the range of 40-50 nm. It's important to note that the glovebox housing the evaporator and the glovebox housing the spincoater were two different gloveboxes.

For proper placement of the substrates within the evaporator's chamber, I utilized a specialized substrate holder composed of a stainless steel frame and a shadow mask, on which substrates were resting during the evaporation process.

Evaporation shadow mask

To evaporate the contact in the specific pattern, the appropriate shadow mask, which is shown in Figure 6.5, was employed to generate a top contact pattern. This pattern is almost identical to the etching pattern but with a 90-degree rotation. Consequently, the actual devices exist as vertical stacks at the intersections of FTO and the deposited metal. After the evaporation process, the samples were sealed using epoxy and microscope



(a) **(b)**

Figure 6.5: **a)** Evaporation shadow mask for top electrode deposition for 4 substrates. The coloured regions are stainless steel, which creates a shadow pattern. A triangle is placed in the top-right corner of each substrate to identify individual devices. **b)** Schematic combination of evaporation and etching patterns. The evaporation pattern is similar to the FTO etching pattern but rotated 90 degrees. The actual devices are sandwiched between the crossing of the leads in the centre of the substrate. Each substrate contains 8 individual devices.

glass slides to prolong their life span by inhibiting atmosphere-related degradation of the Perovskite layer.

To create the shadow mask, the shadow pattern was cut from a stainless steel sheet using the same laser etcher that was also used for FTO etching, namely Rofin EasyMark Laser marker model IV F20. Since the material was different, the laser settings were different as well, and were as follows:

- 8.25 mm Z-axis spacing
- 20 kHz Pulse frequency
- 100 mm / s laser scanning speed
- 70% pumping power
- 0.01 line width

Initially, this laser etcher was not intended for cutting metal sheets, hence it lacked the required power to cut the pattern in one go. Therefore, the marking process was repeated around 50 times using the said parameters.

6.4 Perovskite Memristor Characterisation

To characterise my memristors, I created a dedicated characterisation kit. In this section, I will provide further information regarding the characterisation kit and the procedures involved in characterisation.

6.4.1 Characterisation kit

In order to effectively analyse and understand the electrodynamic properties of my devices, a specialized instrument kit was developed and put together. This kit's overall design is illustrated in Figure 6.6.

At the heart of this setup lies the Ossila Source Measure Unit X200 (SMU) [164], a tool that carries out the essential measurements and captures the current-voltage characteristics of the devices in question. However, one limitation of the Ossila system is that it features only two channels, which means that it can concurrently measure data from just two devices. Beyond this point, a manual rewiring process becomes necessary. To overcome this constraint, I devised a practical solution. Given that my substrates encompassed eight devices each, I devised a straightforward auxiliary circuit. This circuit permits the Ossila to be rerouted to the other devices on the same substrate using relays.

Additionally, a custom sample holder was designed and produced. This holder has the capability to accommodate two substrates simultaneously. Each device present within the substrates is individually linked to the relay board. The configuration of this board dictates which specific device is actively connected to an Ossila channel at any given time. To control the relay board, an Arduino Mega board was employed. This microcontroller is readily available commercially and is easily programmable. Both the Ossila unit and the Arduino board were linked to the central PC via a serial port connection. This connection allowed for seamless coordination of their functions and the efficient management of more complex characterization tasks.

The pivotal software that took charge of controlling both the Ossila system and the Arduino board was realized using LabVIEW [165]. LabVIEW is a graphical programming

environment designed with a focus on scientific and engineering applications. This choice of software facilitated streamlined control and efficient manipulation of the instrumentation for my characterization endeavours.

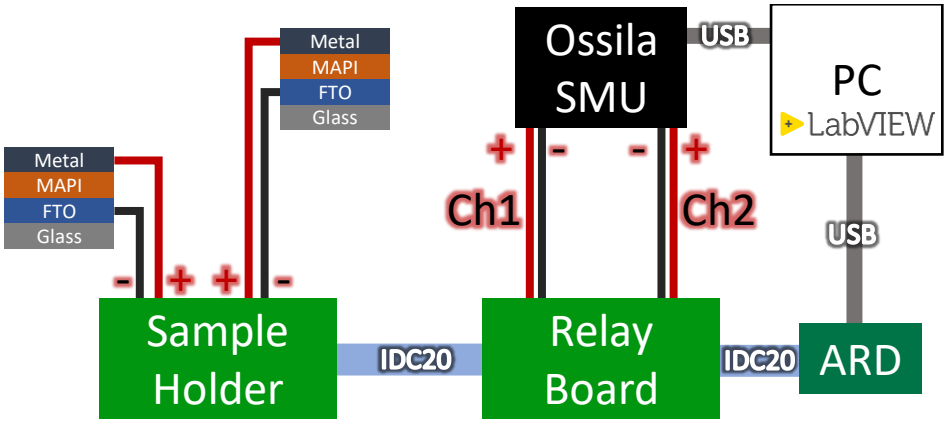


Figure 6.6: Schematic diagram of the memristor characterisation setup. The JV curves are recorded by Ossila SMU, which connects to the individual devices on the substrates via the Arduino-controlled relay board. Both Ossila and Arduino communicate with LabVIEW controlling program via serial connection. The measurement and instrument control are implemented with the LabVIEW program.

Sample holder

The setup for holding the samples comprised a custom-designed PCB, a sample nest, and a holding box. The PCB was created with the purpose of housing and connecting the samples for characterization. To accommodate the substrates, a specialized nest was fabricated using 3D printing. This nest was designed to be directly attachable to the PCB, while also including apertures for pins to pass through. The 3D printing process utilized an UltiMaker S3 printer equipped with an AA 0.8 PLA print core. Printing was executed using the default Fast settings, namely setting the layer height to 0.2mm and the infill density to 20%.

The images of both the PCB and the nest model can be observed in Figure 6.7.

In the sample holder, spring-loaded pins were employed to establish the connection between the devices' pins and the remaining components of the characterization circuit. Each individual pin was linked to the rest of the circuit through the utilization of an

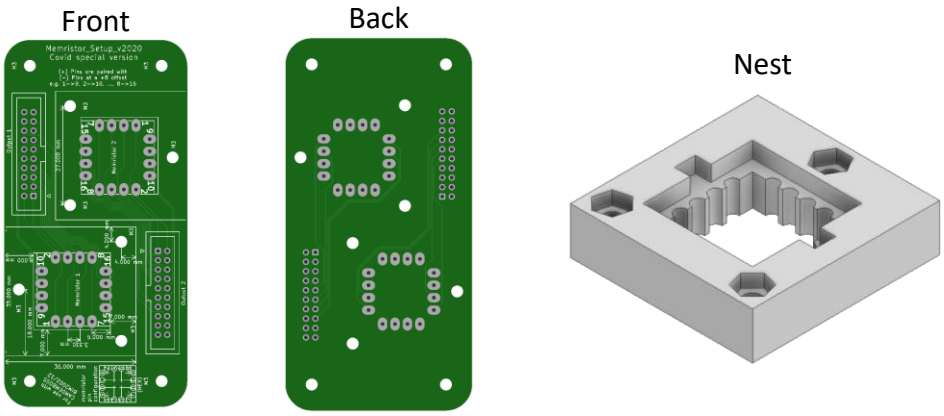


Figure 6.7: **a)** Sample holder's PCB, designed to connect each individual device on a substrate to a pin on IDC20 connection. **b)** Substrate nest, designed to hold the samples securely. Round perforations around the main window allow for the pins to go from PCB to the substrate.

Insulation-displacement connector or IDC for short. Here, in particular, IDC20 ribbon cable was utilised, as illustrated in Figure 6.6.

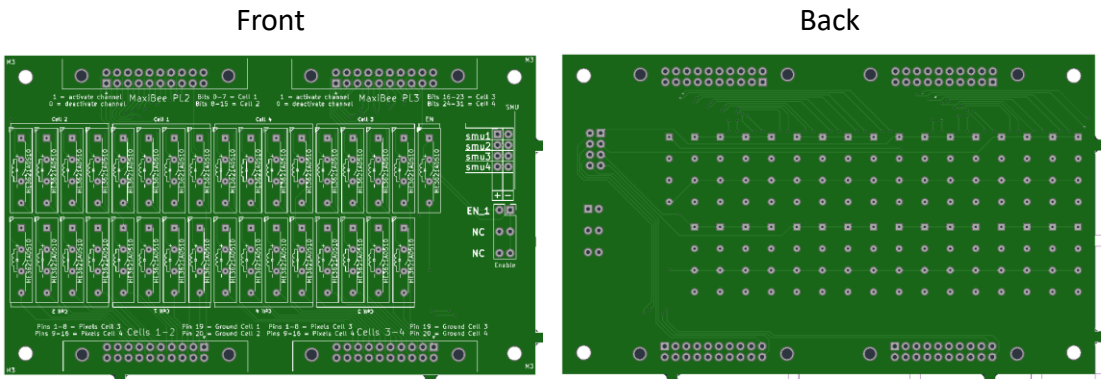
To enhance structural stability and complete the sample holder arrangement, the PCB was affixed onto a plastic box, namely BIM2002/12-GY/GY Plastic enclosure available at Farnell UK. This measure ensured the durability and reliability of the overall setup.

Relay Board

The relay board is responsible for linking the sample holder to the Ossila Source Measure Unit (SMU). This board facilitates the connection of different devices on the same substrate to the SMU without necessitating rewiring.

The board features IDC20 connectors for both the links to the sample holder and the Arduino Mega board, which manages the activation of the relays. This arrangement streamlines the process of managing relay states.

For a visual representation of the board's configuration, please refer to Figure 6.8, showcasing the board schematic. The key element of the board is the HE3621A0510 Reed relay manufactured by Littelfuse. It can be purchased from various distributors, including UK Farnell where I acquired mine.



(a) **(b)**

Figure 6.8: Relay board PCB which connects Ossila SMU channels with the sample holder. The precise configuration of all relays dictates which device on which substrate is connected to the source meter. The board is composed of HE3621A0510 Reed relays. **a)** Front of PCB. **b)** Back of PCB.

6.4.2 LabVIEW controlling program

LabVIEW is a visual programming environment designed with a specific focus on controlling instruments and systems [165]. The choice to use LabVIEW was primarily driven by its user-friendly approach to interfacing with external devices and effortlessly creating graphical interfaces. Figure 6.9 presents the graphical user interface, or GUI, of this program.

During a typical characterization session, the initial step involves establishing connections with the instruments via serial connections, namely Ossial SMU and Arduino. For managing the Ossila SMU, I developed a custom LabVIEW instrument library. This library streamlined the handling of the measurement procedures by utilising pre-built lower-level subfunctions, provided by Ossila themselves. An example of such a procedure schematic is presented in Figure 6.10 simplifying the process of sending commands and fetching data.

In the case of communicating with Arduino, I leveraged an accessible firmware known as LINX. This firmware was loaded onto the Arduino, enabling LabVIEW to easily control it as if it were just another serial-based instrument. Beyond issuing commands and receiving responses from the instruments, LabVIEW's role extended to translating received

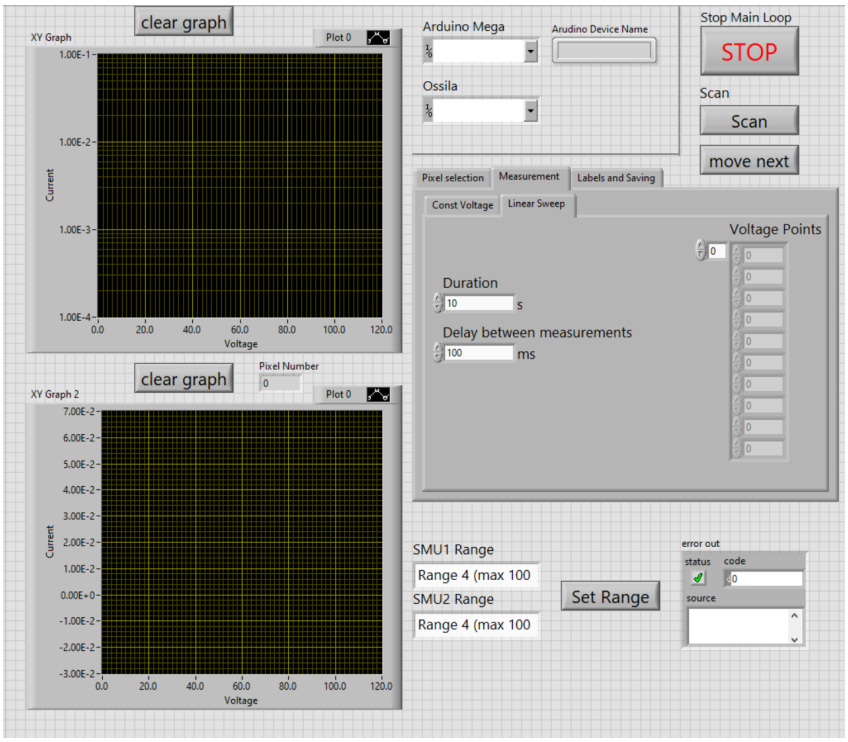


Figure 6.9: Graphical interface of LabVIEW controlling program. The program allows connecting to the instruments and administering current-voltage measurements. The program can measure two devices at the same time and present the JV curve as the measurement goes along.

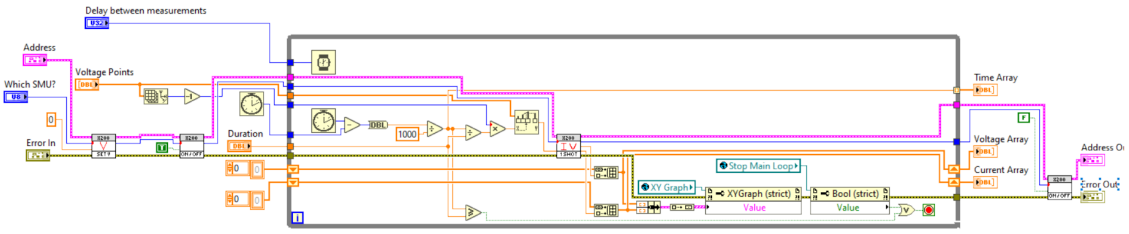


Figure 6.10: Block diagram of the linear sweep measurement for a single device, implemented in LabVIEW. This procedure commands the SMU to apply piece-wise linear bias to a single device of choice.

data, generating plots, and ultimately storing the collected data on the computer for later analysis.

6.4.3 Characterisation procedure

The characterization procedure was carried out within standard indoor lighting conditions, namely under incandescent light. The primary mode of measurement employed was a

piece-wise linear voltage sweep, abbreviated as PWL. Such a sweep is predefined by values stored within the 'Voltage Points' array, visually accessible in Figure 6.9. Such control configuration offered considerable flexibility during the characterization, allowing for real-time adjustments of the applied bias based on the already-collected data visualized in the XY graphs.

To provide a representative example, Figure 6.11 illustrates a typical applied voltage curve. While the choice of this profile might seem arbitrary at first glance, it is, in fact, the result of manual converging, aimed at achieving the maximum possible level of hysteresis for a memristor. By applying such bias, I managed to capture the best performance, that a device could produce.

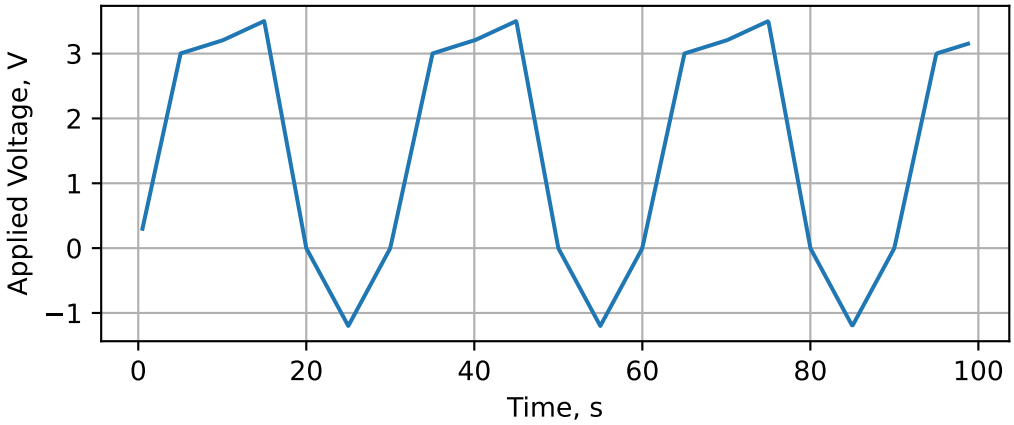


Figure 6.11: Voltage vs. time curve of a typically applied bias to the Halide Perovskite memristor. The curve is piece-wise linear, i.e. it consists of connected linear segments.

6.5 LTspice specifics

This section provides additional details pertaining to the usage of LTspice [142]. Here, it presents important details on accuracy settings. These settings were used to establish stability and practicality in simulations of my Perovskite memristor model, owing to its hybrid nature. Additionally, this section describes the ideal switches which were used in several of the Fuzlearn circuits.

6.5.1 Simulation accuracy

Typically, LTspice's default simulation parameters provide a level of accuracy that meets the demands of most applications satisfactorily. However, there are specific scenarios, such as when dealing with my memristor model characterized by its hybrid behaviour and the real-time solution of non-linear differential equations, where insufficient precision can lead to inaccuracies that manifest as artefacts within the simulation results.

Addressing this concern, I sought solutions online, ultimately discovering a set of parameters that address the precision issue. The original source of these parameters can be found at the following link [166]. These parameters are presented in the format depicted in the table below:

Table 6.1: List of LTspice accuracy enhancing parameters with detailed descriptions of what each one does.

LTspice command	What it does
.OPTIONS maxord=1	Switches the solver's integration method
.OPTIONS numdgt=99	Increases the number of digits in the output file (forces the use of double precision for calculations)
.OPTIONS measdgt=99	
.OPTIONS itl1=1000	Increases the maximum number of iterations in a single calculation, default is 100
.OPTIONS itl2=1000	
.OPTIONS itl4=1000	
.OPTIONS vntol=1e-12	Decreases absolute and relative voltage error tolerances
.OPTIONS reltol=1e-6	
.OPTIONS gmin=1e-15	Decreases the conductance added to every non-linear circuit element for "better convergence"
.OPTIONS abstol=1e-15	Decreases absolute current error tolerance
.OPTIONS plotwinsize=0	Disables the waveform downsampling, i.e. forces the solver to use fixed time deltas

I adopted the specific settings detailed in Table 6.1 for all my LTspice simulations. A notable adjustment within this parameter set entails transitioning from the default 32-bit floating-point representation to a higher precision of 64-bit floating-point numbers.

This transition enables increased numerical accuracy, inhibiting the propagation of errors throughout the simulation process. In addition, the parameter list includes further adjustments that address other limitations imposed by LTspice. This enhances the simulation's flexibility, resulting in a more precise and realistic depiction of memristor behaviour and the circuits they form. By utilizing these customized settings, I managed to achieve a higher level of accuracy in simulating the memristors and their derived circuits.

6.5.2 Switches

As an integral aspect of my circuit designs, the incorporation of switches was imperative. I made a conscious decision to abstain from employing existing switching components and instead opted for an abstract switch approach. By doing so, I refrained from committing to any specific type of switching device, thereby preserving the overarching generality of the hardware system.

Within any logical system, the process of decision-making hinges upon resolving conditional "if statements." In the context of hardware, this functionality is commonly achieved through the utilization of switches. The physical world presents a diverse array of switch options, ranging from basic relays to more advanced elements like MOSFETs and Bipolar transistors, each possessing distinct traits and application scenarios. However, as previously mentioned, the core objective of my project did not revolve around the creation of a tangible, commercially-ready product. Rather, my primary aim was to showcase the potential utility of memristors and explore their hypothetical implementation within hardware. As a result, my emphasis wasn't on generating a fully operational commercial commodity, but rather on substantiating the conceptual framework and validating its feasibility.

To achieve this objective, the switches integrated into my circuits and simulations were intentionally designed to be straightforward and theoretical in nature. Their definitions in LTspice can be observed in the upper part of Figure 6.12(a). These definitions are realised using what are known as 'dot commands' [167]. Specifically, the `.model` command establishes a new model.

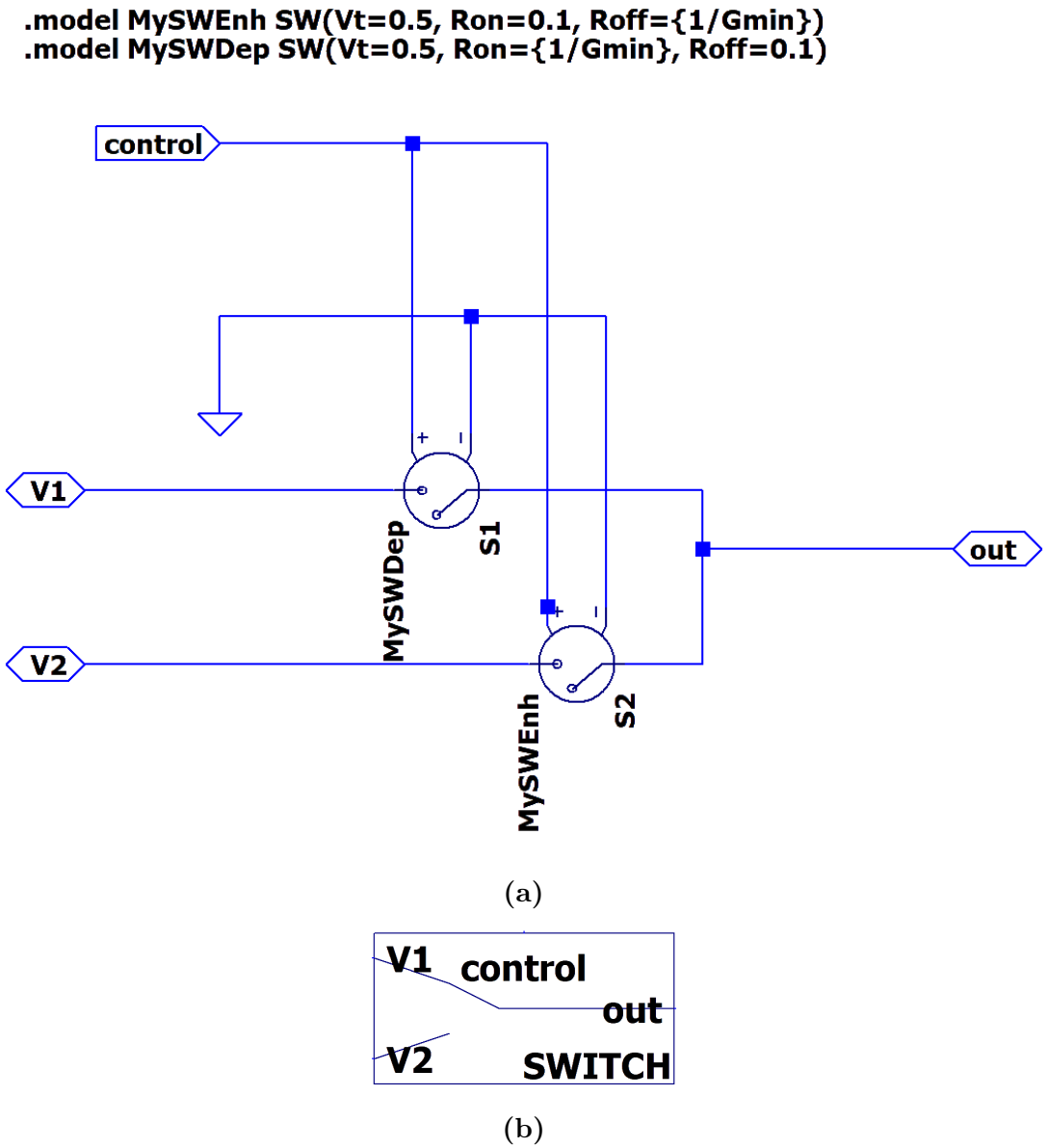


Figure 6.12: LTspice schematic of ideal switches. **a)** Top - LTspice dot commands defining the models. The commands define two models: one for the enhancing mode switch - default: off, powered: on, and another for the depletion mode switch - default: on, powered: off. The schematic in the middle represents the circuit of the two-way switch. Depending on the value of the voltage at **control** pin, pin **out** is connected to **V!** or **V2**. By default, **out** is connected to **V1**. **b)** Subcircuit symbol, which encapsulates the two-way switch subcircuit.

These switches functioned as placeholders, capable of being substituted with real-world counterparts for practical realization. These switches are defined to have similar characteristics, including the same impedances and threshold voltages. Both exhibited a

minimum resistance of 0.1 Ω and a maximum of 10^{15} Ω. The switching action occurred at the identical threshold voltage of 0.5 V. The sole distinction between them lay in their operational modes. One switch operates in *enhancement* mode, necessitating voltage application to establish the connection between contacts, transitioning from high to low resistance. Meanwhile, the other switch functions in *depletion* mode, demanding voltage to sever the connection between contacts, transitioning from low to high resistance.

As depicted in the schematic within Figure 6.12(a), I further streamlined the development of the equivalent Fuzlearn circuit by coupling two of these switches together, creating a two-way switch. This arrangement, instead of merely opening and closing a connection, enables the toggling of the connection between two distinct pathways. This specific subcircuit proved invaluable in encapsulating the Automaton, where its role is to act as a separator between READ and WRITE lines, fostering the facilitation of the circuit's functionality.

6.6 Miscellaneous

Throughout my PhD project, I made use of various instruments for different tasks. This section primarily highlights the tools that aided me in scientific writing, including this very thesis.

6.6.1 Grammarly

I extensively made use of Grammarly [168], a widely employed application for examining grammar and syntax within natural languages, with a particular focus on English. This tool proved to be immensely beneficial, especially considering that English is not my native language.

6.6.2 ChatGPT and Grammarly AI

At the beginning of my thesis writing, I found ChatGPT 3.5 [169] to be an invaluable aid. In general, ChatGPT is a Large Language Model (LLM) [170, 171], a neural network

designed to be an artificial interlocutor, which can be asked to perform various types of tasks from programming aid [172] to holding a casual conversation [173]. In the particular context of this thesis, I used ChatGPT to convert bullet points, with which I expressed my ideas and findings, into a regular type of text. Additionally, once generated, I would review and edit the resulting text since it could contain inaccuracies or misrepresentations of my initial thoughts. Such workflow significantly increased my writing productivity.

However, later on, I found ChatGPT's generated text to be a bit 'flowery', overliterary, and sometimes even pretentious and obnoxious. This was, perhaps, due to ChatGPT's generality, as it was not designed specifically to be a writing aid. To overcome ChatGPT's disadvantages and keep a steady workflow, I adopted another newly released AI writing aid - Grammarly AI, formerly known as GrammarlyGO [174]. Developed by the same company behind Grammarly, this language model shares similarities with ChatGPT, being a chat-driven interlocutor. However, it distinguishes itself by being purpose-built as a writing aid, with a primary focus on text refinement and structural adjustments. I personally found Grammarly AI's generated text to be more modest, clear, and requiring less editing and post-processing.

I fully acknowledge that language models could be employed for content and idea generation. In light of that, I declare that I used the above tools solely within their writing-aid capabilities and that all final ideas, presented in this thesis, are of my own cognitive and intellectual labour.

Chapter 7

Conclusion and Final notes

In the pursuit of solving a distinctive and relatively unexplored challenge, my PhD project focused on enabling in-situ machine learning – the process of executing machine learning tasks within the environment where the input data originates. This endeavour was particularly complex due to the resource-constrained nature of such environments, characterized by limited energy and computational capabilities. Consequently, my research necessitated a highly multidisciplinary approach, encompassing machine learning, material science, and electrical engineering.

Chapter 3 introduced a novel approach to machine learning, which I named Fuzlearn. This architecture, inspired by the Tsetlin Machine, aimed to establish a direct linkage between analogue inputs and transparent decision-making. Unlike conventional neural networks, Fuzlearn sought to surmount the limitations of complex models by offering a streamlined approach to decision-making in in-situ machine learning scenarios.

Chapter 4 explored the potential of a novel electrical component, the Halide Perovskite memristor, as a cornerstone for future-generation machine learning acceleration hardware, which also includes Fuzlearn. Investigating the underlying resistive switching mechanism, I identified an optimal configuration and developed a simplistic yet robust model grounded in first-principles physics. This model offered a versatile framework for circuit design, bridging the gap between theory and application.

Chapter 5 marked the convergence of my research, wherein I employed my Perovskite

memristor model to design a hypothetical Fuzlearn hardware circuit. This circuit, while elegantly straightforward, served as a potent proof of concept, demonstrating the viability of my novel approach to in-situ machine learning. Moreover, this hardware design provides a solid foundation for future expansion and refinement, aiming to transition from a conceptual demonstration to a robust industrial-grade solution.

Looking ahead, my research has laid the groundwork for an array of promising avenues for further exploration. Regarding the theoretical aspect of machine learning, exploring refinements in Fuzlearn's operational protocol stands as a potential avenue for enhancing its efficiency and performance. As of now, this is a clear bottleneck which constrains Fuzlearn's applicability to problems of high dimensionality.

Regarding the Perovskite memristor modelling, my memristor model can be subjected to testing against more involved experiments with parameter variation, such as varying device thickness, L, in both the model and the experiment. Future work can validate the model in its connection to material physics. As such, there is a potential for the model to be utilised for direct mapping of material/device parameters to the actual electro-dynamical behaviour that said device would produce.

Regarding circuit hardware work, the next potential step involves implementing the Fuzlearn hardware circuit in the real world, transitioning from simulation to physical realization. This endeavour would entail selecting suitable peripheral components, ensuring compatibility, and validating their efficacy in practice. Alternatively, the circuit's potential could be stretched further by applying its simulation to more complex problems, such as tackling more intricate datasets like the Iris dataset, etc. This expansion would provide even further insights into the circuit's versatility and its ability to address progressively challenging tasks.

In essence, my PhD project has carved a distinctive path in the domain of in-situ machine learning by bridging disparate disciplines and culminating in the creation of Fuzlearn, a novel machine learning architecture empowered by Halide Perovskite memristors. The confluence of theoretical work and practical implementations highlights not only the project's accomplishments but also its potential to catalyse future research and innovation

at the crossroads of machine learning, material science, and circuit design.

Bibliography

[1] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, in *Machine Learning* (Elsevier, 1983), pp. 3–23, ISBN 9780080510545, URL https://linkinghub.elsevier.com/retrieve/pii/B9780080510545500054.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, Proceedings of the IEEE **107**, 1738 (2019), ISSN 0018-9219, 1558-2256, URL https://ieeexplore.ieee.org/document/8736011/.

[3] J. Almutairi and M. Aldossary, Computers, Materials & Continua **68**, 4143 (2021), ISSN 1546-2226, URL https://www.techscience.com/cmc/v68n3/42528.

[4] B. F. Cornea, A.-C. Orgerie, and L. Lefevre, in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)* (IEEE, Luxembourg, Luxembourg, 2014), pp. 143–148, ISBN 9781479927302, URL http://ieeexplore.ieee.org/document/6968983/.

[5] D. Chen and H. Zhao, in *2012 International Conference on Computer Science and Electronics Engineering* (2012), vol. 1, pp. 647–651.

[6] U. S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley, in *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)* (IEEE, Larnaca, 2017), pp. 1–8, ISBN 9781538637319, URL http://ieeexplore.ieee.org/document/8316459/.

[7] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris, Future Internet **11**, 94 (2019), ISSN 1999-5903, URL https://www.mdpi.com/1999-5903/11/4/94.

[8] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui (2020), URL `https://arxiv.org/abs/2003.12172`.

[9] B. Müller, J. Reinhardt, and M. T. Strickland, *Neural networks: an introduction* (Springer Science & Business Media, 1995).

[10] K. Morik and P. Marwedel, eds., *Machine learning under resource constraints. Volume 1: Fundamentals / edited by Katharina Morik and Peter Marwedel*, vol. 1 of *De Gruyter STEM* (De Gruyter, Berlin Boston, 2023), ISBN 978-3-11-078594-4 978-3-11-078593-7.

[11] M. J. M. Pelgrom, in *Analog-to-Digital Conversion* (Springer New York, New York, NY, 2013), pp. 325–418, ISBN 9781461413707 9781461413714, URL `http://link.springer.com/10.1007/978-1-4614-1371-4_8`.

[12] Y. Zhang, P. Tino, A. Leonardis, and K. Tang, IEEE Transactions on Emerging Topics in Computational Intelligence **5**, 726 (2021), ISSN 2471-285X, URL `https://ieeexplore.ieee.org/document/9521221/`.

[13] W. J. Von Eschenbach, Philosophy & Technology **34**, 1607 (2021), ISSN 2210-5433, 2210-5441, URL `https://link.springer.com/10.1007/s13347-021-00477-0`.

[14] O.-C. Granmo (2018), publisher: arXiv Version Number: 15, URL `https://arxiv.org/abs/1804.01508`.

[15] K. C. Klement, in *Internet Encyclopedia of Philosophy* (2004).

[16] K. D. Abeyrathna, H. S. G. Pussewalage, S. N. Ranasinghe, V. A. Oleshchuk, and O.-C. Granmo, in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)* (IEEE, Canberra, ACT, Australia, 2020), pp. 1121–1130, ISBN 978-1-72812-547-3, URL `https://ieeexplore.ieee.org/document/9308206/`.

[17] K. D. Abeyrathna, B. Bhattarai, M. Goodwin, S. R. Gorji, O.-C. Granmo, L. Jiao, R. Saha, and R. K. Yadav, in *Proceedings of the 38th International Conference*

on Machine Learning, edited by M. Meila and T. Zhang (PMLR, 2021), vol. 139 of *Proceedings of Machine Learning Research*, pp. 10–20, URL https://proceedings.mlr.press/v139/abeyrathna21a.html.

[18] T. Rahman, A. Wheeldon, R. Shafik, A. Yakovlev, J. Lei, O.-C. Granmo, and S. Das, in *2022 International Symposium on the Tsetlin Machine (ISTM)* (IEEE, Grimstad, Norway, 2022), pp. 29–36, ISBN 978-1-66547-116-9, URL https://ieeexplore.ieee.org/document/9923830/.

[19] J. Von Neumann, IEEE Annals of the History of Computing **15**, 27 (1993), ISSN 1058-6180, URL http://ieeexplore.ieee.org/document/238389/.

[20] J. Backus, Communications of the ACM **21**, 613 (1978), ISSN 0001-0782, 1557-7317, URL https://dl.acm.org/doi/10.1145/359576.359579.

[21] W. A. Wulf and S. A. McKee, ACM SIGARCH Computer Architecture News **23**, 20 (1995), ISSN 0163-5964, URL https://dl.acm.org/doi/10.1145/216585.216588.

[22] B. Jacob, S. W. Ng, and D. T. Wang, *Memory systems: cache, DRAM, disk* (Morgan Kaufmann Publishers, Burlington, MA, 2008), ISBN 978-0-12-379751-3, oCLC: ocn154760293.

[23] A. Mahesri and V. Vardhan, in *Power-Aware Computer Systems*, edited by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, et al. (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005), vol. 3471, pp. 165–180, ISBN 978-3-540-29790-1 978-3-540-31485-1, series Title: Lecture Notes in Computer Science, URL http://link.springer.com/10.1007/11574859_12.

[24] L. Chua and Sung Mo Kang, Proceedings of the IEEE **64**, 209 (1976), ISSN 0018-9219, URL http://ieeexplore.ieee.org/document/1454361/.

[25] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L.-Y. Chen, B. Zhang, and

P. Deaville, IEEE Solid-State Circuits Magazine **11**, 43 (2019), ISSN 1943-0582, 1943-0590, URL https://ieeexplore.ieee.org/document/8811809/.

[26] X. Xiao, J. Hu, S. Tang, K. Yan, B. Gao, H. Chen, and D. Zou, Advanced Materials Technologies **5**, 1900914 (2020), ISSN 2365-709X, 2365-709X, URL https://onlinelibrary.wiley.com/doi/10.1002/admt.201900914.

[27] C. Li, X. Lu, W. Ding, L. Feng, Y. Gao, and Z. Guo, Acta Crystallographica Section B Structural Science **64**, 702 (2008), ISSN 0108-7681, URL https://scripts.iucr.org/cgi-bin/paper?S0108768108032734.

[28] M. A. Green, A. Ho-Baillie, and H. J. Snaith, Nature Photonics **8**, 506 (2014), ISSN 1749-4885, 1749-4893, URL https://www.nature.com/articles/nphoton.2014.134.

[29] R. S. Sanchez, V. Gonzalez-Pedro, J.-W. Lee, N.-G. Park, Y. S. Kang, I. Mora-Sero, and J. Bisquert, The Journal of Physical Chemistry Letters **5**, 2357 (2014), ISSN 1948-7185, 1948-7185, URL https://pubs.acs.org/doi/10.1021/jz5011187.

[30] J. You, Z. Hong, Y. M. Yang, Q. Chen, M. Cai, T.-B. Song, C.-C. Chen, S. Lu, Y. Liu, H. Zhou, et al., ACS Nano **8**, 1674 (2014), ISSN 1936-0851, 1936-086X, URL https://pubs.acs.org/doi/10.1021/nn406020d.

[31] X. Zhu, Q. Wang, and W. D. Lu, Nature Communications **11**, 2439 (2020), ISSN 2041-1723, URL https://www.nature.com/articles/s41467-020-16261-1.

[32] I. Shmarov, P. Docampo, T. Billam, and R. Shafik, in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (IEEE, Glasgow, UK, 2020), pp. 1–4, ISBN 978-1-72816-044-3, URL https://ieeexplore.ieee.org/document/9294929/.

[33] I. Shmarov, P. Docampo, T. Billam, R. Shafik, and A. Yakovlev, in *2022 International Symposium on the Tsetlin Machine (ISTM)* (IEEE, Grimstad, Norway,

2022), pp. 89–96, ISBN 978-1-66547-116-9, URL https://ieeexplore.ieee.org/document/9923824/.

[34] S. Suthaharan, in *Machine Learning Models and Algorithms for Big Data Classification* (Springer US, Boston, MA, 2016), vol. 36, pp. 237–269, ISBN 978-1-4899-7640-6 978-1-4899-7641-3, series Title: Integrated Series in Information Systems, URL https://link.springer.com/10.1007/978-1-4899-7641-3_10.

[35] T. Dietterich, ACM Computing Surveys **27**, 326 (1995), ISSN 0360-0300, 1557-7341, URL https://dl.acm.org/doi/10.1145/212094.212114.

[36] S. W. Menard, *Applied logistic regression analysis*, no. no. 07-106 in Sage university papers. Quantitative applications in the social sciences (Sage Publications, Thousand Oaks, Calif, 2002), 2nd ed., ISBN 978-0-7619-2208-7.

[37] L. E. Peterson, Scholarpedia **4**, 1883 (2009).

[38] I. Steinwart and A. Christmann, *Support vector machines*, Information science and statistics (Springer, New York, NY, 2008), 1st ed., ISBN 978-0-387-77242-4 978-0-387-77241-7 978-1-4899-8963-5.

[39] H. Núñez, C. Angulo, and A. Català (????).

[40] N. Barakat and A. P. Bradley, Neurocomputing **74**, 178 (2010), ISSN 09252312, URL https://linkinghub.elsevier.com/retrieve/pii/S0925231210001591.

[41] H. Adeli, Computer-Aided Civil and Infrastructure Engineering **16**, 126 (2001), publisher: Wiley Online Library.

[42] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Pattern Recognition **77**, 354 (2018), ISSN 00313203, URL https://linkinghub.elsevier.com/retrieve/pii/S0031320317304120.

[43] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. (2021), publisher: arXiv Version Number: 1, URL https://arxiv.org/abs/2103.00020.

[44] OpenAI (2023), publisher: arXiv Version Number: 3, URL https://arxiv.org/abs/2303.08774.

[45] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers* (Cambridge University Press, 2021), 1st ed., ISBN 978-1-108-86060-4 978-1-108-49493-9, URL https://www.cambridge.org/core/product/identifier/9781108860604/type/book.

[46] T. Cetto, J. Byrne, X. Xu, and D. Moloney, in *Recent Advances in Big Data and Deep Learning*, edited by L. Oneto, N. Navarin, A. Sperduti, and D. Anguita (Springer International Publishing, Cham, 2020), vol. 1, pp. 17–26, ISBN 978-3-030-16840-7 978-3-030-16841-4, series Title: Proceedings of the International Neural Networks Society, URL http://link.springer.com/10.1007/978-3-030-16841-4_3.

[47] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean (2021), publisher: arXiv Version Number: 3, URL https://arxiv.org/abs/2104.10350.

[48] L. Baischer, M. Wess, and N. TaheriNejad (2021), publisher: arXiv Version Number: 1, URL https://arxiv.org/abs/2104.09252.

[49] E. Strubell, A. Ganesh, and A. McCallum (2019), publisher: arXiv Version Number: 1, URL https://arxiv.org/abs/1906.02243.

[50] K. Lu, P. Mardziel, F. Wu, P. Amancharla, and A. Datta, in *Logic, Language, and Security*, edited by V. Nigam, T. Ban Kirigin, C. Talcott, J. Guttman, S. Kuznetsov, B. Thau Loo, and M. Okada (Springer International Publishing, Cham, 2020), vol. 12300, pp. 189–202, ISBN 978-3-030-62076-9 978-3-030-62077-6, series Title: Lecture Notes in Computer Science, URL http://link.springer.com/10.1007/978-3-030-62077-6_14.

[51] A. H. Sham, K. Aktas, D. Rizhinashvili, D. Kuklianov, F. Alisinanoglu, I. Ofodile, C. Ozcinar, and G. Anbarjafari, Signal, Image and Video Processing **17**, 399

(2023), ISSN 1863-1703, 1863-1711, URL https://link.springer.com/10.1007/s11760-022-02246-8.

[52] B. Lwowski and A. Rios, Journal of the American Medical Informatics Association **28**, 839 (2021), ISSN 1527-974X, URL https://academic.oup.com/jamia/article/28/4/839/6114713.

[53] A. Hart and J. Wyatt, Medical Informatics **15**, 229 (1990), ISSN 0307-7640, URL http://www.tandfonline.com/doi/full/10.3109/14639239009025270.

[54] W. N. Price, Science Translational Medicine **10**, eaao5333 (2018), ISSN 1946-6234, 1946-6242, URL https://www.science.org/doi/10.1126/scitranslmed.aao5333.

[55] B. Goodman and S. Flaxman (2016), publisher: arXiv Version Number: 3, URL https://arxiv.org/abs/1606.08813.

[56] M. Kissner (2019), publisher: arXiv Version Number: 2, URL https://arxiv.org/abs/1911.07658.

[57] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, IEEE Access **7**, 47230 (2019), ISSN 2169-3536, URL https://ieeexplore.ieee.org/document/8685687/.

[58] M. L. Tsetlin, **22**, 1345 (1961).

[59] J. Lei, A. Wheeldon, R. Shafik, A. Yakovlev, and O.-C. Granmo, in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (IEEE, Glasgow, UK, 2020), pp. 1–4, ISBN 978-1-72816-044-3, URL https://ieeexplore.ieee.org/document/9294877/.

[60] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O.-C. Granmo, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **378**, 20190593 (2020), ISSN 1364-503X, 1471-2962, URL https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0593.

[61] G. T. Berge, O.-C. Granmo, T. O. Tveit, M. Goodwin, L. Jiao, and B. V. Matheussen, IEEE Access **7**, 115134 (2019), ISSN 2169-3536, URL `https://ieeexplore.ieee.org/document/8798633/`.

[62] K. D. Abeyrathna, O.-C. Granmo, X. Zhang, and M. Goodwin, in *Advances and Trends in Artificial Intelligence. From Theory to Practice*, edited by F. Wotawa, G. Friedrich, I. Pill, R. Koitz-Hristov, and M. Ali (Springer International Publishing, Cham, 2019), vol. 11606, pp. 564–578, ISBN 978-3-030-22998-6 978-3-030-22999-3, series Title: Lecture Notes in Computer Science, URL `https://link.springer.com/10.1007/978-3-030-22999-3_49`.

[63] K. D. Abeyrathna, O.-C. Granmo, X. Zhang, and M. Goodwin, in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)* (IEEE, Canberra, ACT, Australia, 2020), pp. 2084–2092, ISBN 978-1-72812-547-3, URL `https://ieeexplore.ieee.org/document/9308291/`.

[64] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface* (Morgan Kaufmann Publishers, Burlington, MA, 2009), 4th ed., ISBN 978-0-12-374493-7.

[65] Y. Zhang and S. Swanson, in *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)* (IEEE, Santa Clara, CA, USA, 2015), pp. 1–10, ISBN 978-1-4673-7619-8, URL `http://ieeexplore.ieee.org/document/7208275/`.

[66] A. Eliahu, R. Ronen, P.-E. Gaillardon, and S. Kvatinsky, ACM Journal on Emerging Technologies in Computing Systems **17**, 1 (2021), ISSN 1550-4832, 1550-4840, URL `https://dl.acm.org/doi/10.1145/3432815`.

[67] D. Rossi, I. Loi, F. Conti, G. Tagliavini, A. Pullini, and A. Marongiu, in *2014 IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)* (IEEE, Eilat, Israel, 2014), pp. 1–5, ISBN 978-1-4799-5988-4 978-1-4799-5987-7, URL `http://ieeexplore.ieee.org/document/7005803/`.

[68] L. Chua, IEEE Transactions on Circuit Theory **18**, 507 (1971), ISSN 0018-9324, URL http://ieeexplore.ieee.org/document/1083337/.

[69] S. P. Adhikari, M. P. Sah, H. Kim, and L. O. Chua, IEEE Transactions on Circuits and Systems I: Regular Papers **60**, 3008 (2013), ISSN 1549-8328, 1558-0806, URL http://ieeexplore.ieee.org/document/6549211/.

[70] Y. Ho, G. M. Huang, and P. Li, in *Proceedings of the 2009 International Conference on Computer-Aided Design* (ACM, San Jose California, 2009), pp. 485–490, ISBN 978-1-60558-800-1, URL https://dl.acm.org/doi/10.1145/1687399.1687491.

[71] Y. V. Pershin and M. Di Ventra, Advances in Physics **60**, 145 (2011), ISSN 0001-8732, 1460-6976, URL http://www.tandfonline.com/doi/abs/10.1080/00018732.2010.544961.

[72] Y. Chen, IEEE Transactions on Electron Devices **67**, 1420 (2020), ISSN 0018-9383, 1557-9646, URL https://ieeexplore.ieee.org/document/8961211/.

[73] D. Marković, A. Mizrahi, D. Querlioz, and J. Grollier, Nature Reviews Physics **2**, 499 (2020), ISSN 2522-5820, URL https://www.nature.com/articles/s42254-020-0208-2.

[74] J.-W. Jang, S. Park, Y.-H. Jeong, and H. Hwang, in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)* (IEEE, Melbourne VIC, Australia, 2014), pp. 1054–1057, ISBN 978-1-4799-3432-4 978-1-4799-3431-7, URL http://ieeexplore.ieee.org/document/6865320/.

[75] G. Indiveri, E. Linn, and S. Ambrogio, in *Resistive Switching*, edited by D. Ielmini and R. Waser (Wiley, 2016), pp. 715–736, 1st ed., ISBN 978-3-527-33417-9 978-3-527-68087-0, URL https://onlinelibrary.wiley.com/doi/10.1002/9783527680870.ch25.

[76] S. Park, A. Sheri, J. Kim, J. Noh, J. Jang, M. Jeon, B. Lee, B. R. Lee, B. H. Lee, and H. Hwang, in *2013 IEEE International Electron Devices Meeting* (IEEE,

Washington, DC, USA, 2013), pp. 25.6.1–25.6.4, ISBN 978-1-4799-2306-9, URL http://ieeexplore.ieee.org/document/6724692/.

[77] J. M. Azpiroz, E. Mosconi, J. Bisquert, and F. De Angelis, Energy & Environmental Science **8**, 2118 (2015), ISSN 1754-5692, 1754-5706, URL http://xlink.rsc.org/?DOI=C5EE01265A.

[78] C. Eames, J. M. Frost, P. R. F. Barnes, B. C. O'Regan, A. Walsh, and M. S. Islam, Nature Communications **6**, 7497 (2015), ISSN 2041-1723, URL https://www.nature.com/articles/ncomms8497.

[79] N. J. Jeon, J. H. Noh, Y. C. Kim, W. S. Yang, S. Ryu, and S. I. Seok, Nature Materials **13**, 897 (2014), ISSN 1476-1122, 1476-4660, URL https://www.nature.com/articles/nmat4014.

[80] N. Duraisamy, N. M. Muhammad, H.-C. Kim, J.-D. Jo, and K.-H. Choi, Thin Solid Films **520**, 5070 (2012), ISSN 00406090, URL https://linkinghub.elsevier.com/retrieve/pii/S0040609012002441.

[81] L. Qiu, S. He, L. K. Ono, S. Liu, and Y. Qi, ACS Energy Letters **4**, 2147 (2019), ISSN 2380-8195, 2380-8195, URL https://pubs.acs.org/doi/10.1021/acsenergylett.9b01396.

[82] W. Tress, The Journal of Physical Chemistry Letters **8**, 3106 (2017), ISSN 1948-7185, 1948-7185, URL https://pubs.acs.org/doi/10.1021/acs.jpclett.7b00975.

[83] A. R. Bowring, L. Bertoluzzi, B. C. O'Regan, and M. D. McGehee, Advanced Energy Materials **8**, 1702365 (2018), ISSN 16146832, URL https://onlinelibrary.wiley.com/doi/10.1002/aenm.201702365.

[84] X. Zhu, J. Lee, and W. D. Lu, Advanced Materials **29**, 1700527 (2017), ISSN 0935-9648, 1521-4095, URL https://onlinelibrary.wiley.com/doi/10.1002/adma.201700527.

[85] K. Yan, M. Peng, X. Yu, X. Cai, S. Chen, H. Hu, B. Chen, X. Gao, B. Dong, and D. Zou, Journal of Materials Chemistry C **4**, 1375 (2016), ISSN 2050-7526, 2050-7534, URL `http://xlink.rsc.org/?DOI=C6TC00141F`.

[86] J. C. Pérez-Martínez, M. Berruet, C. Gonzales, S. Salehpour, A. Bahari, B. Arredondo, and A. Guerrero, Advanced Functional Materials p. 2305211 (2023), ISSN 1616-301X, 1616-3028, URL `https://onlinelibrary.wiley.com/doi/10.1002/adfm.202305211`.

[87] J. S. Han, Q. V. Le, J. Choi, K. Hong, C. W. Moon, T. L. Kim, H. Kim, S. Y. Kim, and H. W. Jang, Advanced Functional Materials **28**, 1705783 (2018), ISSN 1616301X, URL `https://onlinelibrary.wiley.com/doi/10.1002/adfm.201705783`.

[88] K. S. Narendra and M. A. L. Thathachar, *Learning automata: an introduction* (Prentice Hall, Englewood Cliffs, N.J, 1989), ISBN 978-0-13-485558-5.

[89] A. Rezvanian, A. M. Saghiri, S. M. Vahidipour, M. Esnaashari, and M. R. Meybodi, *Recent Advances in Learning Automata*, vol. 754 of *Studies in Computational Intelligence* (Springer International Publishing, Cham, 2018), ISBN 978-3-319-72427-0 978-3-319-72428-7, URL `http://link.springer.com/10.1007/978-3-319-72428-7`.

[90] P. Cunningham, M. Cord, and S. J. Delany, in *Machine Learning Techniques for Multimedia*, edited by M. Cord and P. Cunningham (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), pp. 21–49, ISBN 978-3-540-75170-0 978-3-540-75171-7, series Title: Cognitive Technologies, URL `http://link.springer.com/10.1007/978-3-540-75171-7_2`.

[91] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, Computer science (McGraw-Hill, 2009), ISBN 9780262533058.

[92] T. E. Oliphant, Computing in Science & Engineering **9**, 10 (2007), ISSN 1521-9615, URL `http://ieeexplore.ieee.org/document/4160250/`.

[93] D. Dua and C. Graff, *UCI machine learning repository* (2019).

[94] R. A. Fisher, *Iris* (1936), URL `https://archive.ics.uci.edu/dataset/53`.

[95] H. Pistori and J. J. Neto, CLEI Electronic Journal **6** (2018), ISSN 0717-5000, URL `http://www.clei.org/cleiej/index.php/cleiej/article/view/349`.

[96] J. P. Pinto, S. Kelur, and J. Shetty, in *2018 4th International Conference for Convergence in Technology (I2CT)* (IEEE, Mangalore, India, 2018), pp. 1–4, ISBN 978-1-5386-5232-9, URL `https://ieeexplore.ieee.org/document/9057891/`.

[97] K. F. Hussain, M. Y. Bassyouni, and E. Gelenbe, *Accurate and Energy-Efficient Classification with Spiking Random Neural Network: Corrected and Expanded Version* (2019), arXiv:1906.08864 [cs, stat], URL `http://arxiv.org/abs/1906.08864`.

[98] S. Rahimi Gorji, O.-C. Granmo, A. Phoulady, and M. Goodwin, in *Artificial Intelligence XXXVI*, edited by M. Bramer and M. Petridis (Springer International Publishing, Cham, 2019), vol. 11927, pp. 146–151, ISBN 978-3-030-34884-7 978-3-030-34885-4, series Title: Lecture Notes in Computer Science, URL `http://link.springer.com/10.1007/978-3-030-34885-4_11`.

[99] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles, Monthly Notices of the Royal Astronomical Society **459**, 1104 (2016), ISSN 0035-8711.

[100] E. Fortunato, D. Ginley, H. Hosono, and D. C. Paine, MRS Bulletin **32**, 242 (2007), ISSN 1938-1425, 0883-7694, URL `https://www.cambridge.org/core/journals/mrs-bulletin/article/abs/transparent-conducting-oxides-for-photovoltaics/F5F6FB985D7CCB418EE3487F33AB3FEC`.

[101] J. Y. Kim, J.-W. Lee, H. S. Jung, H. Shin, and N.-G. Park, Chemical Reviews

120, 7867 (2020), ISSN 0009-2665, 1520-6890, URL https://pubs.acs.org/doi/10.1021/acs.chemrev.0c00107.

[102] A. E. Rakhshani, Y. Makdisi, and H. A. Ramazaniyan, Journal of Applied Physics **83**, 1049 (1998), ISSN 0021-8979, 1089-7550, URL https://pubs.aip.org/jap/article/83/2/1049/530643/Electronic-and-optical-properties-of-fluorine.

[103] S.-F. Tseng, W.-T. Hsiao, K.-C. Huang, and D. Chiang, Applied Surface Science **257**, 8813 (2011), ISSN 01694332, URL https://linkinghub.elsevier.com/retrieve/pii/S0169433211005915.

[104] P. S. Whitfield, N. Herron, W. E. Guise, K. Page, Y. Q. Cheng, I. Milas, and M. K. Crawford, Scientific Reports **6**, 35685 (2016), ISSN 2045-2322, URL https://www.nature.com/articles/srep35685.

[105] D. W. Ferdani, S. R. Pering, D. Ghosh, P. Kubiak, A. B. Walker, S. E. Lewis, A. L. Johnson, P. J. Baker, M. S. Islam, and P. J. Cameron, Energy & Environmental Science **12**, 2264 (2019), ISSN 1754-5692, 1754-5706, URL http://xlink.rsc.org/?DOI=C9EE00476A.

[106] W. Shockley, Bell System Technical Journal **28**, 435 (1949), ISSN 00058580, URL https://ieeexplore.ieee.org/document/6773080.

[107] R. T. Tung, Applied Physics Reviews **1**, 011304 (2014), ISSN 1931-9401, URL https://pubs.aip.org/apr/article/1/1/011304/123905/The-physics-and-chemistry-of-the-Schottky-barrier.

[108] S. M. Sze, *Physics of semiconductor devices* (Wiley, New York, 1981), 2nd ed., ISBN 9780471056614.

[109] P. Delugas, C. Caddeo, A. Filippetti, and A. Mattoni, The Journal of Physical Chemistry Letters **7**, 2356 (2016), ISSN 1948-7185, 1948-7185, URL https://pubs.acs.org/doi/10.1021/acs.jpclett.6b00963.

[110] A. Khalid, A. M. Alanazi, S. A. Alderhami, A. H. Alsehli, M. M. Alsowayigh, A. M. Saeedi, H. B. Albargi, and H. M. Al-Saidi, Materials Science in Semiconductor Processing **165**, 107678 (2023), ISSN 13698001, URL https://linkinghub.elsevier.com/retrieve/pii/S1369800123003712.

[111] U. Ali, K. J. B. A. Karim, and N. A. Buang, Polymer Reviews **55**, 678 (2015), ISSN 1558-3724, 1558-3716, URL http://www.tandfonline.com/doi/full/10.1080/15583724.2015.1031377.

[112] B. Li, Y. Li, C. Zheng, D. Gao, and W. Huang, RSC Advances **6**, 38079 (2016), ISSN 2046-2069, URL https://pubs.rsc.org/en/content/articlelanding/2016/ra/c5ra27424a.

[113] N. Semaltianos, Microelectronics Journal **38**, 754 (2007), ISSN 00262692, URL https://linkinghub.elsevier.com/retrieve/pii/S0026269207000870.

[114] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, Nature **453**, 80 (2008), ISSN 0028-0836, 1476-4687, URL https://www.nature.com/articles/nature06932.

[115] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, Journal of Applied Physics **106**, 074508 (2009), ISSN 0021-8979, 1089-7550, URL https://pubs.aip.org/jap/article/106/7/074508/896021/Switching-dynamics-in-titanium-dioxide-memristive.

[116] J. G. Simmons, Journal of Applied Physics **34**, 1793 (1963), ISSN 0021-8979, 1089-7550, URL https://pubs.aip.org/jap/article/34/6/1793/362794/Generalized-Formula-for-the-Electric-Tunnel-Effect.

[117] V. Gupta, G. Lucarelli, S. Castro-Hermosa, T. Brown, and M. Ottavi, Microelectronics Reliability **111**, 113708 (2020), ISSN 00262714, URL https://linkinghub.elsevier.com/retrieve/pii/S002627141930839X.

[118] R. Millikan, E. Bishop, and A. T. Society, *Elements of Electricity: A Practical Discussion of the Fundamental Laws and Phenomena of Electricity and Their Practical Applications in the Business and Industrial World* (American Technical Society, 1917), URL `https://books.google.co.uk/books?id=dZM3AAAAMAAJ`.

[119] T. Williams, *The circuit designer's companion* (Newnes, Amsterdam Paris, 2005), 2nd ed., ISBN 978-0-7506-6370-0.

[120] D. P. Hampshire (2015), publisher: arXiv Version Number: 2, URL `https://arxiv.org/abs/1510.04309`.

[121] S. Selberherr, *Analysis and Simulation of Semiconductor Devices* (Springer Vienna, Vienna, 1984), ISBN 978-3-7091-8754-8 978-3-7091-8752-4, URL `http://link.springer.com/10.1007/978-3-7091-8752-4`.

[122] P. Farrell, N. Rotundo, D. H. Doan, M. Kantner, J. Fuhrmann, and T. Koprucki (2016), medium: application/pdf Publisher: Berlin : Weierstraß-Institut für Angewandte Analysis und Stochastik, URL `https://oa.tib.eu/renate/handle/123456789/2481`.

[123] D. J. Griffiths, *Introduction to electrodynamics* (Prentice Hall, Upper Saddle River, N.J, 1999), 3rd ed., ISBN 978-0-13-805326-0.

[124] M. S. Branicky, in *Handbook of Networked and Embedded Control Systems*, edited by D. Hristu-Varsakelis and W. S. Levine (Birkhäuser Boston, Boston, MA, 2005), pp. 91–116, ISBN 978-0-8176-3239-7 978-0-8176-4404-8, URL `http://link.springer.com/10.1007/0-8176-4404-0_5`.

[125] E. W. Dijkstra, *Selected writings on computing: a personal perspective*, Texts and monographs in computer science (Springer-Verlag, New York, 1982), ISBN 978-0-387-90652-2.

[126] J. G. Reich, in *Kinetic Data Analysis*, edited by L. Endrenyi (Springer US, Boston,

MA, 1981), pp. 39–50, ISBN 978-1-4613-3257-2 978-1-4613-3255-8, URL http://link.springer.com/10.1007/978-1-4613-3255-8_3.

[127] R. Goebel, R. G. Sanfelice, and A. R. Teel, IEEE Control Systems **29**, 28 (2009), ISSN 1066-033X, 1941-000X, URL https://ieeexplore.ieee.org/document/4806347/.

[128] R. Mitchell and I. of Electrical Engineers, eds., *Managing complexity in software engineering*, no. 17 in IEE computing series (Peregrinus on behalf of the Institution of Electrical Engineers, Hitchin, 1990), ISBN 978-0-86341-171-7, oCLC: ocm21598451.

[129] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al., Nature Methods **17**, 261 (2020), ISSN 1548-7091, 1548-7105, URL https://www.nature.com/articles/s41592-019-0686-2.

[130] J. Dormand and P. Prince, Journal of Computational and Applied Mathematics **6**, 19 (1980), ISSN 03770427, URL https://linkinghub.elsevier.com/retrieve/pii/0771050X80900133.

[131] Scipy, *solve_ivp*, URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html.

[132] Scipy, *square*, URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.square.html.

[133] A. Törn and A. Žilinskas, eds., *Global Optimization*, vol. 350 of *Lecture Notes in Computer Science* (Springer Berlin Heidelberg, Berlin, Heidelberg, 1989), ISBN 978-3-540-50871-7 978-3-540-46103-6, URL https://link.springer.com/10.1007/3-540-50871-6.

[134] Scipy, *dual_annealing*, URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.dual_annealing.html.

[135] C. Tsallis and D. A. Stariolo, Physica A: Statistical Mechanics and its Applications **233**, 395 (1996), ISSN 03784371, URL https://linkinghub.elsevier.com/retrieve/pii/S0378437196002713.

[136] Y. Xiang, D. Sun, W. Fan, and X. Gong, Physics Letters A **233**, 216 (1997), ISSN 03759601, URL https://linkinghub.elsevier.com/retrieve/pii/S037596019700474X.

[137] P. J. M. Van Laarhoven and E. H. L. Aarts, in *Simulated Annealing: Theory and Applications* (Springer Netherlands, Dordrecht, 1987), pp. 7–15, ISBN 978-90-481-8438-5 978-94-015-7744-1, URL http://link.springer.com/10.1007/978-94-015-7744-1_2.

[138] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Science **220**, 671 (1983), ISSN 0036-8075, 1095-9203, URL https://www.science.org/doi/10.1126/science.220.4598.671.

[139] W. Shockley, The Bell System Technical Journal **30**, 990 (1951).

[140] C. Bi, Y. Shao, Y. Yuan, Z. Xiao, C. Wang, Y. Gao, and J. Huang, J. Mater. Chem. A **2**, 18508 (2014), ISSN 2050-7488, 2050-7496, URL http://xlink.rsc.org/?DOI=C4TA04007D.

[141] D.-H. Kwon, K. M. Kim, J. H. Jang, J. M. Jeon, M. H. Lee, G. H. Kim, X.-S. Li, G.-S. Park, B. Lee, S. Han, et al., Nature Nanotechnology **5**, 148 (2010), ISSN 1748-3387, 1748-3395, URL https://www.nature.com/articles/nnano.2009.456.

[142] M. T. Engelhardt, *LTspice*, URL https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html.

[143] L. W. Nagel and D. Pederson, Tech. Rep. UCB/ERL M382, EECS Department, University of California, Berkeley (1973), URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html.

[144] LTspice, *Arbitrary behaviour Resistor*, URL https://ltwiki.org/index.php?title=Undocumented_LTspice#Behavioral_Resistors.

[145] G. L. Zeng and M. Zeng, in *Electric Circuits* (Springer International Publishing, Cham, 2021), pp. 43–48, ISBN 978-3-030-60514-8 978-3-030-60515-5, URL http://link.springer.com/10.1007/978-3-030-60515-5_7.

[146] Numpy, *Numpy*, URL https://numpy.org/.

[147] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al., Nature **585**, 357 (2020), ISSN 0028-0836, 1476-4687, URL https://www.nature.com/articles/s41586-020-2649-2.

[148] T. E. Oliphant and others, *Guide to numpy*, vol. 1 (Trelgol Publishing USA, 2006).

[149] MATLAB, *MATLAB*, URL https://uk.mathworks.com/products/matlab.html.

[150] J. Ranjani, A. Sheela, and K. P. Meena, in *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)* (IEEE, CHENNAI, India, 2019), pp. 1–5, ISBN 978-1-72811-604-4, URL https://ieeexplore.ieee.org/document/8741475/.

[151] Scipy, *Scipy*, URL https://scipy.org/.

[152] Scipy, *Scipy-Integration*, URL https://docs.scipy.org/doc/scipy/tutorial/integrate.html.

[153] Scipy, *Scipy-Interpolation*, URL https://docs.scipy.org/doc/scipy/tutorial/interpolate.html.

[154] Matplotlib, *Maptlotlib*, URL https://matplotlib.org/.

[155] Matplotlib, *Pyplot*, URL https://matplotlib.org/stable/tutorials/pyplot.html.

[156] B. P. Pokkunuri, ACM SIGPLAN Notices **24**, 96 (1989), ISSN 0362-1340, 1558-1160, URL https://dl.acm.org/doi/10.1145/71605.71612.

[157] Python, *Dictionaries*, URL https://docs.python.org/3/tutorial/datastructures.html#dictionaries.

[158] Python, *Multiprocessing*, URL https://docs.python.org/3/library/multiprocessing.html.

[159] Python, *Pool*, URL https://docs.python.org/3/library/multiprocessing.html#multiprocessing.pool.Pool.

[160] I. Shmarov, *Fuzlearn Python Implementation* (2022), URL https://github.com/IvanShmarov/fuzlearn.

[161] Hellma, *Hellmanex® III*, URL https://www.hellma.com/en/laboratory-supplies/cuvettes/hellmanex-cleaning-concentrate/.

[162] Ossila, *UV Ozone Cleaner*, URL https://www.ossila.com/products/uv-ozone-cleaner.

[163] R. G. Larson and T. J. Rehg, in *Liquid Film Coating*, edited by S. F. Kistler and P. M. Schweizer (Springer Netherlands, Dordrecht, 1997), pp. 709–734, ISBN 978-94-010-6246-6 978-94-011-5342-3, URL http://link.springer.com/10.1007/978-94-011-5342-3_20.

[164] Ossila, *Source Measure Unit*, URL https://www.ossila.com/products/xtralien-source-measure-unit-source-meter.

[165] NI, *Labview*, URL https://www.ni.com/en-gb/shop/product/labview.html.

[166] turingbirds, *LTspice accuracy settings* (2017), URL https://gist.github.com/turingbirds/c90672c3b126d0d5f37f90494d5057cb.

[167] LTspice, *Dot Commands*, URL https://ltwiki.org/LTspiceHelpXVII/LTspiceHelp/html/DotCommands.htm.

[168] Grammarly, *Grammarly*, URL `https://www.grammarly.com/`.

[169] OpenAI, *ChatGPT*, URL `https://chat.openai.com/`.

[170] OpenAI, *Better language models and their implications* (2019), URL `https://openai.com/research/better-language-models`.

[171] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. (2022), publisher: arXiv Version Number: 2, URL `https://arxiv.org/abs/2206.07682`.

[172] N. M. S. Surameery and M. Y. Shakor, International Journal of Information technology and Computer Engineering pp. 17–22 (2023), ISSN 2455-5290, URL `http://journal.hmjournals.com/index.php/IJITC/article/view/1679/1993`.

[173] W. Zhao, Y. Zhao, X. Lu, S. Wang, Y. Tong, and B. Qin (2023), publisher: arXiv Version Number: 1, URL `https://arxiv.org/abs/2304.09582`.

[174] Grammarly, *Grammarly AI*, URL `https://www.grammarly.com/ai`.