SpikeExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA

*electronics*

MDPI

*Article*

# SpikeExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA

**Dario Padovano, Alessio Carpegna, Alessandro Savino and Stefano Di Carlo ***

Control and Computer Engineering Department, Politecnico di Torino, 10129 Torino, Italy; dario.padovano@studenti.polito.it (D.P.); alessio.carpegna@polito.it (A.C.); alessandro.savino@polito.it (A.S.)
* Correspondence: stefano.dicarlo@polito.it

**Abstract:** One of today's main concerns is to bring artificial intelligence capabilities to embedded systems for edge applications. The hardware resources and power consumption required by state-of-the-art models are incompatible with the constrained environments observed in edge systems, such as IoT nodes and wearable devices. Spiking Neural Networks (SNNs) can represent a solution in this sense: inspired by neuroscience, they reach unparalleled power and resource efficiency when run on dedicated hardware accelerators. However, when designing such accelerators, the amount of choices that can be taken is huge. This paper presents SpikExplorer, a modular and flexible Python tool for hardware-oriented Automatic Design Space Exploration to automate the configuration of FPGA accelerators for SNNs. SpikExplorer enables hardware-centric multiobjective optimization, supporting target factors such as accuracy, area, latency, power, and various combinations during the exploration process. The tool searches the optimal network architecture, neuron model, and internal and training parameters leveraging Bayesian optimization, trying to reach the desired constraints imposed by the user. It allows for a straightforward network configuration, providing the full set of explored points for the user to pick the trade-off that best fits their needs. The potential of SpikExplorer is showcased using three benchmark datasets. It reaches 95.8% accuracy on the MNIST dataset, with a power consumption of 180 mW/image and a latency of 0.12 ms/image, making it a powerful tool for automatically optimizing SNNs.

**Keywords:** neuromorphic; Spiking Neural Networks; hardware accelerators; FPGA; Design Space Exploration; network architecture search; hyperparameter optimization



**Citation:** Padovano, D.; Carpegna, A.; Savino, A.; Di Carlo, S. SpikExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA. *Electronics* **2024**, *13*, 1744. https://doi.org/10.3390/electronics13091744

Academic Editors: Gerard Ghibaudo and Francis Balestra

Received: 4 April 2024
Revised: 27 April 2024
Accepted: 29 April 2024
Published: 1 May 2024



## 1. Introduction

The field of Artificial Intelligence (AI), particularly of Artificial Neural Networks (ANNs), proliferates, with different solutions tailored for diverse computational tasks. In the plethora of available ANN models, we can include Multi Layer Perceptrons (MLPs) that are well suited for pattern recognition; Recurrent Neural Networks (RNNs), such as Long Short Term Memory (LSTM) that can efficiently process time series, Convolutional Neural Networks (CNNs) for image analysis, and Transformers for Natural Language Processing (NLP). Amidst this variety, Spiking Neural Networks (SNNs) [1] emerge as a new computing paradigm, shaped by neuroscience models exploring networks of biological neurons [2]. Differently from other types of ANNs, SNNs mimic the behavior of biological neurons more faithfully, trying to reach the extreme energy efficiency observed in our brain. Although this goal is still far, SNNs are already able to outperform State of Art (SoA) ANN models in many different applications, in particular those for which the energy consumption is somewhat constrained [3]. SNNs become particularly interesting when implemented through dedicated hardware co-processors. Indeed, the intrinsic efficiency of these models makes them especially suitable to be implemented on digital Application-Specific Integrated Circuits (ASICs) [4], Field Programmable Gate Arrays (FPGAs) [5], and analog dedicated circuits [6].

In this context, one of the main challenges is determining how to construct the SNN to fit the target application best: there are many different neuron models with varying degrees of biological plausibility and computing efficiency; a single model has a lot of internal parameters to tune; the network architecture itself can be modified depending on the task to perform. A manual selection of all these hyperparameters can be very complex and could bring a nonoptimal solution. At the same time, an exhaustive search for the best configuration would require too much time, given the search space size. Automatic Design Space Exploration (ADSE) can represent a solution. However, while the literature is rich in works about ADSE in the field of CNNs [7,8] and other ANN models [9], this is not true for SNNs. The few existing works on the topic focus on a single-objective optimization directed towards the improvement of the accuracy [10] or concentrate the search on a particular aspect of the network, like the input data encoding, using fixed neuron models and parameters and performing only a tiny grid search between a limited set of network architectures [11].

This paper presents SpikeExplorer, a flexible hardware-oriented ADSE framework to automatically optimize SNN models for their deployment on digital hardware accelerators targeting FPGA implementations. The tool supports multiobjective ADSE driven by power consumption, latency, area, and accuracy, leveraging Bayesian optimization. It empowers users to fine-tune network architecture, neuron models, and internal settings, explicitly tailoring them for FPGA deployment. SpikeExplorer can be specialized for whatever neuron model and hardware implementation, allowing to easily customize SNN co-processors depending on the user requirements. This can help leverage the benefits of SNNs in power and resource-constrained edge applications [12], simplifying the configuration and tuning of these new networks in various problems.

The paper is organized as follows: Section 2 provides the required SNN background and Section 3 overviews related work on ADSE for SNNs. Section 4 overviews the proposed method and Section 5 shows its capabilities on a set of case studies. Finally, Section 6 concludes the paper and highlights future extensions.

## 2. Background

AI is reaching unparalleled performance, matching human capabilities in complex tasks like pattern recognition, NLP, and object detection. However, it still stands orders of magnitude behind human intelligence regarding energy efficiency [13]. When it comes to optimization, nature excels, and its solution to minimize brain power consumption is to make neurons communicate through asynchronous sequences of spikes. SNNs are based on the same communication approach, drawing inspiration from biology to model how neurons react to these spikes. In an SNN, the information is encoded in the timing of the spikes, regarding them as binary events. Therefore, from a computational perspective, neurons in an SNN handle streams of single-bit data, strongly reducing the overall required complexity. Different neuron models can react to spikes in various ways. Neurons can be interconnected differently, and training algorithms can tune the resulting network on a specific problem. This leads to a huge design space that requires proper techniques to be analyzed and reduced. The following sections show a subset of all the possible design choices that can be considered in the search.

### *2.1. Network Architecture*

Neurons can be interconnected in various patterns to construct complex networks. One widely used connection scheme is the Fully-Connected (FC) architecture, which can extract complex features from input data. Neural network connections typically adhere to either a Feed Forward (FF) architecture that facilitates a linear information flow from inputs to outputs or adopts recurrent structures with feedback connections, allowing information to loop back. Figure 1a shows the two alternative architectures. Spiking neurons inherently exhibit recurrence since their state is computed starting from the previous one. Hence, the architecture retains information from previous states even in the context of FF SNNs.

However, explicit feedback connections might be necessary to capture longer dependencies or complex dynamics. SpikeExplorer aims to optimize FC architectures organized in layers, interconnected both in an FF and recurrent manner. These architectures are general enough to address most Machine Learning (ML) problems.

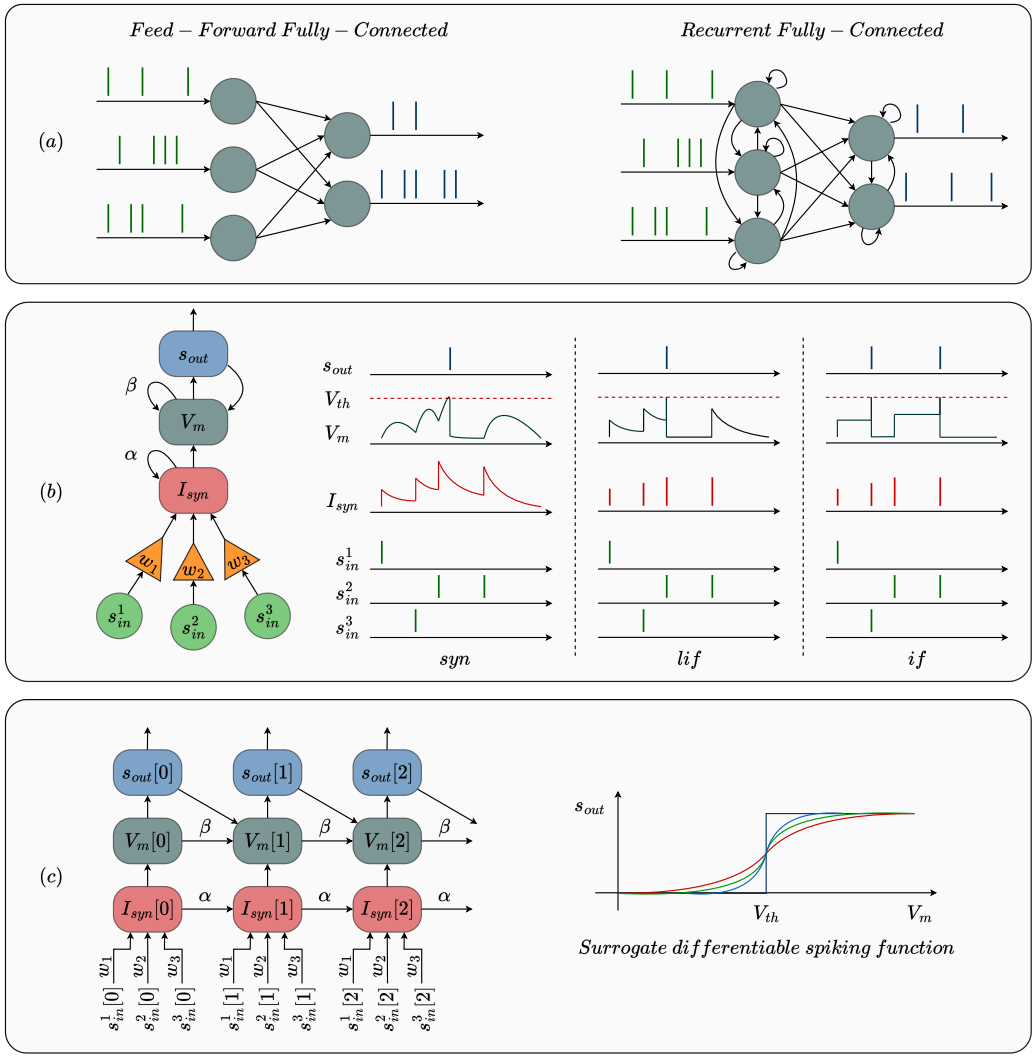



**Figure 1.** SNN Design Space: (**a**) different network architectures; (**b**) different neuron models; (**c**) graph unrolling during training and example of possible surrogate curves to smooth the Heaviside function, replacing it with a differentiable altrernative.

### *2.2. Neuron Models*

The first computational models of biological neurons were developed starting from the accurate observation of electrical propagation inside neural cells [14]. Nonetheless, for practical computational tasks, such a high level of biological fidelity is unnecessary and overly complex. Different simplified alternatives have been developed in the last decades [15]. The most used one is the family of Integrate and Fire (IF) models [16], able to describe neuron dynamics with limited computational complexity. Essentially, an IF neuron functions as an integrator, accumulating spikes over time, and subsequently fires itself a spike if the cumulative value surpasses a predefined threshold. Inputs are transmitted to the neuron via synapses, where they undergo preprocessing before reaching their destination. The most complex IF neuron model is the conductance-based Leaky Integrate and Fire (LIF) model, described, in discrete time, by Equations (1)–(3). The synaptic current generated by the synapse has a dynamic response to the input spikes. The $i$-th synapse weights the input spikes through its synaptic weight $w_i$. Without stimuli, the current decays exponentially

toward a rest value. Equation (1) shows a compact description of the total net synaptic current $I_{syn}$ received by the neuron: each synapse has its weight and the product $W \cdot s_{in}[n]$ represents the weighting operation performed by N synapses on as many inputs. All the synapses share the exponential decay rate $\alpha$ (with $\alpha < 1$).

$$I_{syn}[n] = \alpha \cdot I_{syn}[n-1] + W \cdot s_{in}[n] \tag{1}$$

The input current is then integrated by the neuron into its membrane potential $V_m$. If the result stays below a threshold value $V_{th}$, $V_m$ follows a temporal dynamic similar to the synaptic one, so it decays exponentially with a decay rate $\beta$ (with $\beta < 1$). If, instead, $V_m$ exceeds $V_{th}$, it is reset by the function $R$, and an output spike $s_{out}$ is generated.

$$V_m[n] = \beta \cdot (V_m[n-1] - s_{out}[n-1] \cdot R[n]) + I_{syn}[n] \tag{2}$$

Equation (3) shows two possible mechanisms for the reset operation. In the first case, called hard reset, $V_m$ is always reset to zero when the threshold is exceeded, i.e., when a spike is generated. In the subtractive reset alternative, the threshold is subtracted by $V_m$.

$$\begin{aligned} R_{hard}[n] &= V_m[n-1] \\ R_{sub}[n] &= V_{th} \end{aligned} \qquad s_{out}[n] = \begin{cases} 1, & \text{if } V_m > V_{th} \\ 0, & \text{if } V_m \leq V_{th} \end{cases} \tag{3}$$

In the rest of the paper, what was just described will be called the synaptic model (abbreviated as *syn*), following the terminology used in [17]. The synaptic model can be simplified by removing the dynamic response of the synapse, considering only the synaptic weight, as is generally performed in ANNs. This is equivalent to setting $\alpha = 0$ in Equation (1). The result is a simple LIF model, referenced in the rest of the paper as *lif*. Finally, the neuron's dynamic response could also be neglected, transforming the neuron into a simple integrator with memory. This can be obtained by setting $\beta = 1$. The result is a basic IF model (referred to as *if*). Figure 1b summarizes these three behaviors, showing an example of their temporal response to spikes. Therefore, even considering only the IF family of models, it is clear that a lot of knobs can be tuned during design. For example, $\alpha$ and $\beta$ determine how fast the exponential decay is, influencing the capability of the neuron to keep the memory of past information, while $V_{th}$ and $R$ affect the firing rate of the neuron and the timing of the output spikes.

### *2.3. Training*

Training SNNs remains an active area of research, drawing inspiration from both biological observations [18] and classical supervised approaches in ML. However, a significant challenge arises when using supervised approaches with SNNs due to the nondifferentiable nature of the output spike function $s_{out}$ concerning the neuron's state $V_m$, as illustrated in Equation (2). Consequently, the traditional backpropagation training algorithm [19] and its derivatives are impractical. To address this issue, a commonly adopted approach involves substituting the spike function with a differentiable surrogate during the backward pass [20]. Various options exist for this surrogate, typically encompassing smoothed versions of the step function, such as the sigmoid and its derivatives, arc, or hyperbolic tangents. Once the nondifferentiability is mitigated, training proceeds akin to that of RNNs: the network can be unrolled and trained using the Back-Propagation Through Time (BPTT) algorithm, propagating the output error across both space (layers of the network) and time (unrolled states of the network). Figure 1c shows the unrolling process and a graphical example of surrogate spike functions used during the backward pass. Subsequently, selecting and fine-tuning an appropriate surrogate function and backpropagation parameters, including learning rate, regularization parameters, optimizer settings, etc., are essential steps in the training process.

### *2.4. Automatic Design Space Exploration*

When working with complex systems such as SNNs, the numerous degrees of freedom make a comprehensive exploration of the design space impractical. This challenge is compounded when crafting specific hardware implementations, where synthesizing and simulating architectures can consume significant time. Over the past few decades, researchers have sought ways to optimize and speed up the search for optimal architectures in electronic systems, a pursuit intensified by the proliferation of AI and ANNs. One approach to reducing the search space involves randomly selecting a subset of points and focusing exploration solely on them. Despite its simplicity, this can prove effective in many cases [21]. Nevertheless, structured and systematic alternatives abound in the literature, many drawing inspiration from biological evolution, like evolutionary and genetic algorithms [22], extremal optimization [23], and Reinforcement Learning (RL) [24].

Among optimization techniques, Bayesian optimization [25] stands out as a robust solution, particularly for its ability to converge rapidly even with complex models. It effectively addresses the exploration–exploitation dilemma [26], balancing exploring new solutions and exploiting known ones. Instead of directly interacting with the objective function (e.g., accuracy of the network), which might be computationally expensive to evaluate, Bayesian optimization builds a simpler, approximate model. It is typically based on Gaussian processes or other probabilistic models. Initially, this surrogate model makes some assumptions about the objective function based on the limited information available. As more data points are collected through evaluations of the actual objective, the surrogate model becomes refined and better approximates the actual function. A Bayesian optimizer relies on an acquisition function, considering both exploration and exploitation aspects to decide which point in the search space to evaluate next. Exploration involves trying out points in the search space that are uncertain or have yet to be explored to gain more information about the objective function and potentially discover better solutions. Moreover, exploitation involves focusing on areas of the search space likely to yield good results based on the current knowledge provided by the surrogate model. The acquisition function balances these two aspects to guide the search effectively. Thanks to this methodical approach, Bayesian optimization demonstrates efficacy in converging to solutions, even in scenarios involving numerous parameters in the search, rendering it a valuable tool for optimizing SNNs. Interested readers may refer to [27] for a broader topic overview.

## 3. Related Works

Among the large plethora of ANNs models, SNNs are the ones that mostly require dedicated hardware co-processors. Indeed, SNNs are characterized by high computational parallelism, lightweight communication channels exchanging asynchronous spikes, and co-location of memory and computing. This fits poorly with the Von-Neumann computing paradigm adopted in general-purpose computers, which relies on a limited number of computational units exchanging data and instructions with a centralized memory. Even specialized architectures, such as Graphic Processing Units (GPUs) and Tensor Processing Units (TPUs), optimized for standard ANN workloads, struggle to process SNNs efficiently [28]. Consequently, employing dedicated neuromorphic hardware emerges as the most efficient solution, especially in contexts where efficiency is the primary concern [28].

In this landscape, one of the solutions that is gaining attention is to exploit the reconfigurability of FPGAs to design application-specific FPGA-based SNNs co-processors [29–33]. The advantage of using FPGAs is their intrinsic reconfigurability, which reduces design time and makes network customization easier. This enables the fine-tuning of hardware implementations for SNNs according to specific problem requirements, configuring the hardware to deploy the most optimized solution. Automatic optimization for SNN architectures is critical when considering these dedicated hardware implementations, particularly for resource-constrained edge applications. In such scenarios, the optimization of the network targets multiple objectives: together with the fine-tuning of the model on a specific

problem, minimizing power consumption, area occupancy, and latency become integral parts of the optimization goals.

Within this framework, tools are available to support FPGA hardware designs. For instance, $E^3NE$ [34] provides a library of elementary blocks to build Register Transfer Level (RTL) descriptions of SNN architectures. On the other hand, Spiker+ [33] provides a framework to automatize the generation of the SNN RTL models starting from a high-level network description, providing a library of possible models and network architectures. However, a crucial gap remains: given the availability of various neuron blocks and architectures, how can the network be optimized to achieve the highest possible accuracy while constraining other metrics such as latency, power consumption, or area? Some works on Network Architecture Search (NAS) for SNNs exist. For example, authors in [35] propose an ADSE methodology to perform a single-objective search, targeting the optimization of SNN accuracy only. They mainly focus on convolutional architectures targeting image datasets, such as CIFAR-10, CIFAR-100, and TinyImageNet, applying a NAS strategy to select between different convolutional kernel and pooling sizes. Therefore, the target application is particular, and the work considers the software model only without considering the actual hardware implementation. On the other hand, reference [10] proposes NeuroXplorer, a hardware-oriented ADSE tool to optimize SNN deployment on existing neuromorphic hardware. There is no search for the network structure, neuron model, and parameters. Conversely, starting from a trained SNN model, the tool tries to organize computations to fit the target platform at best, for example, clustering groups of neurons to minimize the transport of spikes over long distances. It focuses on the computational paradigms used within existing neuromorphic processors, such as the Dynap-se1 [36]. Eventually, the first attempt at creating an FPGA-oriented optimizer was performed in [11]. However, the work is focused on finding the optimal encoding for the input data and on the fast evaluation of the optimization metrics (such as power, area, and latency) performed with a novel system C simulator of the hardware accelerator called NAXT. The optimal SNN search is a grid search conducted within a small set of predefined architectures with a fixed IF neuron model without using any specific optimization algorithm.

## 4. Materials and Methods

SpikeExplorer has been designed as a modular Python tool with different components connected in a closed loop. Figure 2 shows a high-level view of the complete framework.

The Design Space Exploration (DSE) engine is the core of the optimization framework. It aims at finding the optimal SNN architecture and its related parameters for a given problem within a user-defined design space. The user imposes constraints by specifying which parameters must remain fixed and which require optimization. An infinite search space risks prolonged search duration and potential converging failure. Hence, users are prompted to define search limits for each optimized parameter. This ensures that the search remains bounded and manageable. For instance, limits can be set on the maximum network size, considering the available hardware resources on the target platform.

The optimization process follows a multiobjective Bayesian approach. The user can select a set of optimization targets: accuracy, area, latency, and power. While Figure 2 illustrates an exploration encompassing all four potential metrics, the optimization can focus solely on a subset, or even just one in the extreme case. Once the optimization objectives are defined, the DSE engine constructs a surrogate model for each of them and starts an iterative optimization process. The surrogate models determine the next point to explore at each iteration, aiming to optimize all required metrics. A point within the search space is defined by a set of values associated with the parameters used for the optimization.

For each explored design option, the specific SNN architecture and configuration is forwarded to the Network Evaluator (NE), responsible for the network construction, training, and performance evaluation. This, in turn, requires providing a training dataset. This block closes the loop by giving the DSE engine the characterization of the selected observation points in terms of accuracy, area, latency, and power required to update the

internal surrogate models. This task requires comprehensively characterizing the various neuron models and computing their individual area occupancy, power consumption, and latency. SpikeExplorer has been intentionally designed to be versatile and compatible with any user-defined neuron characterization model. However, this paper utilizes a comprehensive characterization library derived from open-source experiments, leveraging the Spiker+ framework [33]. Given the framework's complexity, the following sections overview each component separately.

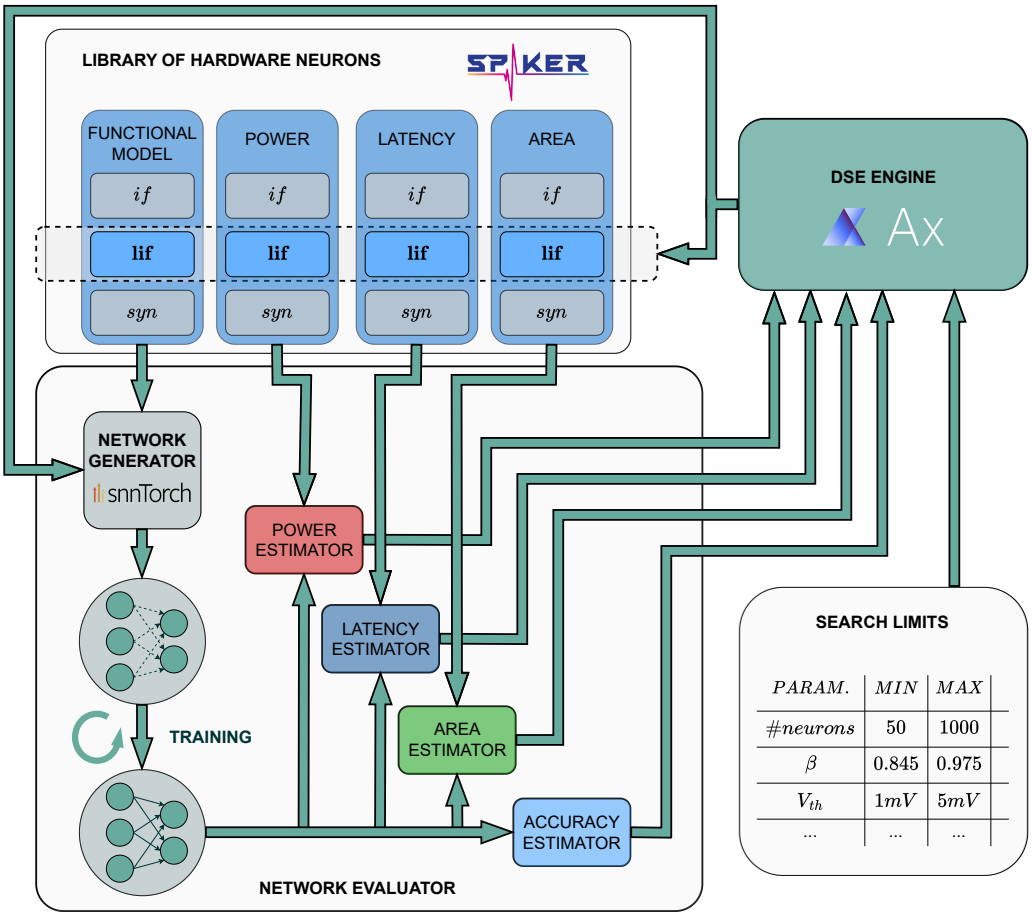



**Figure 2.** SpikeExplorer general architecture, including (i) a library of hardware neurons, (ii) a network evaluator estimating the performance of selected implementations, and (iii) a Bayesian DSE engine.

### *4.1. Network Generator and Hardware Neurons*

The Network Generator (NG) is the submodule of the NE block in charge of building the SNN network models required for performance evaluations. It involves a set of functional models of the available neurons that can be incorporated into the network architecture. Each functional model must be characterizable for the considered optimization targets. For instance, if the optimization target is area minimization, the user must provide a characterization detailing the area occupation of each considered neuron model. This facilitates fine-tuning the search process with specific neuron implementations, which will be integrated into the customized SNN co-processor on FPGA.

In its current implementation, SpikeExplorer supports a set of default neuron functional models based on the IF variants described in Section 2.2. From a functional point of view, the models are defined using the snnTorch framework [17]. This facilitates the creation of a range of networks suitable for various problems where IF models are applicable. snnTorch enables the modeling and approximation of the hardware neuron behavior without a precise knowledge or description of all internal details.

Each available neuron model is associated with hardware-related information obtained using the open-source hardware models provided by the Spiker+ framework [33]. These models are synthesized on a Xilinx XC7Z020 reference FPGA board, and the corresponding performance metrics are extracted, such as area, power, and latency. The following sections outline the techniques to characterize the default SpikeExplorer neuron models. This presentation aims to provide insight into the available estimates and to explain how neurons can be described to fine-tune the search on a specific implementation.

### *4.2. Area*

The neuron area estimation consists of two primary components: (i) the area occupied by the computational elements and (ii) the memory utilized by the synaptic connections. Both are estimated through hardware synthesis of available implementations. Figure 3a shows an example of the synthesis of a simple LIF model, reporting the corresponding Look Up Table (LUT) count and the amount of memory required by synaptic weights, in this case, stored in FPGA Block RAM (BRAM).

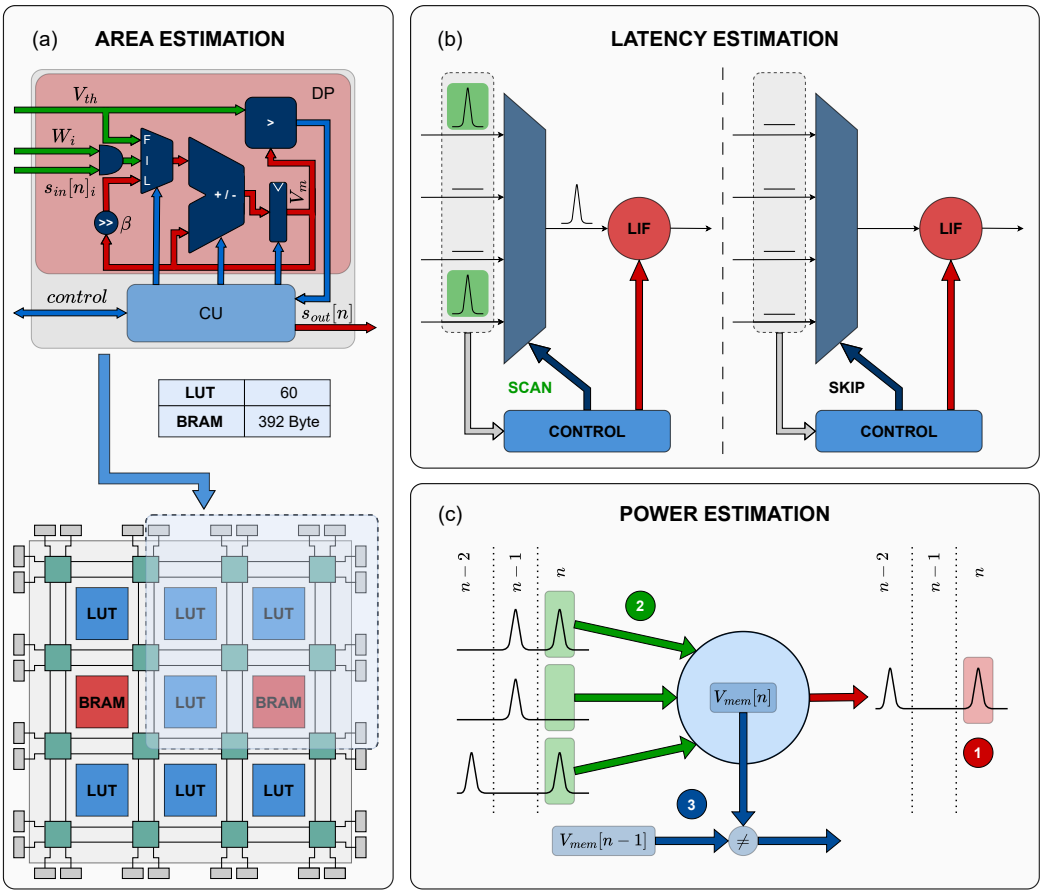



**Figure 3.** Metrics estimation: the figure graphically showcases how the different metrics considered by SpikeExplorer are estimated. (**a**) The area is estimated by synthesizing the target neuron and measuring the required number of LUTs and memory cells; (**b**) optimized latency estimation: if no spike is present in input, no scan is performed, avoiding wasting time; (**c**) the neuron is analyzed to understand its state: (1) if it generated an output spike it means the membrane must be reset, (2) if input spikes are present, they will be integrated into the membrane potential, (3) check if the membrane underwent changes and consider power consumption only if it did.

Quantization is a well-known technique exploited to reduce the memory footprint of SNN models, and several quantization frameworks exist [37–39]. To avoid overlap with existing solutions, the default neuron library provided by SpikeExplorer does not aim at optimizing quantization, focusing on higher-level architectural optimizations. Therefore,

the default neuron characterization uses 32-bit data representations. Experimental results later show that this data representation preserves full-precision accuracy without introducing bias from precision reductions across different models. Although the resulting architecture may appear oversized, what truly influences the optimization process is the relative dimensions of the neurons. Nevertheless, the user can enlarge the library of available neurons, including architectures with different quantization levels, to drive the search toward smaller neurons with more aggressive quantization.

The total number of weights is computed at run-time after defining the SNN architecture and integrated into the area estimation to account for different architectural structures and their impact on the area. This allows us to consider the diverse memory footprints of different architectural choices. For instance, in fully connected architectures with identical neuron count on each layer, a deeper network featuring smaller layers will incorporate fewer synaptic weights, thus necessitating less memory. Moreover, recurrent architectures introduce an area overhead due to FC recurrent connections that can be computed according to Equation (4).

$$R = \frac{Recurrent\ layer\ area}{FF\ layer\ area} = \frac{N_{in} \cdot N_{neurons} + N_{neurons} \cdot N_{neurons}}{N_{in} \cdot N_{neurons}} = \frac{N_{in} + N_{neurons}}{N_{in}} \quad (4)$$

Here, $N_{in}$ denotes the number of inputs, and $N_{neurons}$ signifies the number of neurons within a specific layer.

SpikeExplorer measures the overall area occupancy in terms of Equivalent Look Up Table (ELUT) count:

$$N_{ELUT} = \sum_{l=0}^{N_{layers}} N^{l}_{neurons} \cdot \left( N_{LUT32} + N^{l}_{in} \cdot r \right) \quad (5)$$

where $l$ denotes the layer index, $N_{layers}$ the total number of layers, $N_{ELUT}$ represents the total number of ELUTs occupied by the network, $N_{LUT32}$ is the number of LUTs required by a single neuron with a 32-bit precision, and

$$r = \begin{cases} R, & \text{if l is recurrent} \\ 1, & \text{if l is FF} \end{cases} \quad (6)$$

For clarity in visualization, the distinction between FF and recurrent architectures is expressed using the neuron model nomenclature. The default supported models encompass *if*, *rif*, *lif*, *rlif*, *syn*, and *rsyn*, where the prefix *r* signifies a recurrent architecture.

### *4.3. Accuracy and Latency*

Since quantization is not the primary focus of SpikeExplorer, the accuracy estimation of various network configurations used to drive the DSE process is based on full-precision 64-bit floating-point software models constructed by the NG using the snnTorch framework. These estimations are crucial for guiding the optimization process but should not be regarded as precise accuracy measurements for the target hardware co-processor. They represent an upper bound on the final accuracy that depends on the quantization applied when deploying the model on a real FPGA.

In terms of latency, a clock-driven reference model is considered. In particular, SpikeExplorer implements two different latency estimation models: a fixed latency model, in which each neuron is characterized by a single latency value, independent of the spiking activity, which accounts for the time required to integrate spikes and to decay or reset the neuron; and an optimized latency model, following a computational methodology like that described in [33]. In this case, two latency values are considered: a high latency occurs when at least one input spike is present, prompting neurons to scan all inputs to identify the active ones, and a low value occurs without spikes, where the scanning process is omitted. Figure 3b shows the two considered cases. Since all the inputs are processed se-

quentially, the larger the number of inputs to a neuron, the higher will be the latency of that neuron in case it receives input spikes. The computational process is considered entirely parallel, making the overall latency independent of the overall number of neurons. The approach is highly tailored to fully parallel clock-driven implementations. Alternatively, an activity-based methodology resembling the one utilized for power consumption (refer to Section 4.4) could be adopted to accommodate event-driven approaches.

### *4.4. Power*

The neurons' power consumption generally depends on their activity levels. This holds for clock-driven architectures, as evidenced in [33], and is even more pronounced in event-driven alternatives. To understand how SpikeExplorer estimates the overall power consumption, it is convenient to analyze the operations involved in updating a LIF neuron. Equation (7) shows the mathematical operations involved, obtained by merging Equations (1) and (2), setting $\alpha = 0$ and reordering the terms.

$$V_m[n] = \underbrace{\beta \cdot V_m[n-1]}_{\text{(3) Leak}} + \underbrace{W \cdot s_{in}[n]}_{\text{(2) Integrate}} - \underbrace{\beta \cdot s_{out}[n-1] \cdot R[n]}_{\text{(1) Fire}} \tag{7}$$

As the name of the model suggests, the neuron executes three primary operations: leakage (3), integration (2), and firing (1). The equation defines the evolution of the membrane potential in its discrete-time form. SpikeExplorer examines the state of each neuron at every time step to evaluate the instantaneous power consumption. It expects a characterization of the power consumed by the neuron when executing each of the reported operations. The overall power consumption is then computed by averaging the instantaneous values over the entire sequence of time steps. To accomplish this task, SpikeExplorer must monitor (i) the presence of an output spike, (ii) the presence of input spikes, and (iii) the value of the state variables that change dynamically during the network operations. With LIF and IF models, the only state variable involved is $V_m$, while with a synaptic model, $I_{syn}$ is monitored as well. Using these, SpikeExplorer understands the current state of the neuron and infers the relative consumed power, as illustrated in Figure 3c.

Observing the neuron's output reveals whether the neuron has "fired" a spike. If a spike is generated due to the threshold potential being exceeded, the membrane is reset; this is associated with a first power contribution. Inspecting the inputs, if spikes are present, they are weighted and integrated into the membrane potential, implying an additional power contribution. Eventually, without spikes, the membrane decays toward its resting value, consuming extra power. This condition happens when the membrane potential at time step $n$ differs from that at time step $n-1$. This approach facilitates a highly adaptable evaluation. For instance, in clock-driven update policies, decay consumes power at every time step, which can be factored into the leak contribution. Conversely, in an event-driven approach, computations occur solely in the presence of input spikes, potentially resulting in zero power consumption for the leak term. In this case, the decay power can be merged into the "integrate" term. Alternatively, a custom functional model can be used for the neuron, in which the membrane is updated only when input spikes are received. Here, by checking whether the membrane has changed value, the decay contribution can be considered only in the presence of input stimuli. Lastly, depending on factors like recent resetting or reaching asymptotic decay values due to finite precision platforms, the neuron may remain in a constant state without necessitating significant power-consuming updates.

### *4.5. DSE Engine*

As detailed in Section 2.4, Bayesian optimization emerges as the preferred method for DSE in SNNs. This preference stems from several factors, including the abundance of tunable hyperparameters, inherent noise in the objective function due to the spiking information encoding, and the long training times associated with large SNNs. Bayesian optimization is advantageous for its rapid convergence, facilitated by a simplified surrogate model, and its inherently parallelizable nature, accelerating the exploration process.

The DSE engine of SpikeExplorer is built resorting to the Adaptive eXperimentation (AX) optimization package, an open-source solution developed at Meta™ [40]. It provides high-level Application Programming Interfaces (APIs) that SpikeExplorer uses to iterate through the optimization efficiently. Listing 1 shows a summarized version of the code used to perform the optimization. The DSE engine receives in input a set of configuration parameters, indicating the number of iterations involved in the optimization (line 7), the objectives of the search (line 8), the metrics to optimize, each associated with the range in which to perform the search (line 9), and the set of candidate neuron architectures, including the functional models and their characterization (line 10). The optimization process (lines 12–40) starts initializing the Bayesian surrogate model using the AX APIs (lines 14–20) and then performs an iterative procedure (lines 24–38). A set of parameters is selected at each iteration, following the predictions performed with the surrogate model (line 27). The network is configured with the chosen parameters (line 29), and its performance is evaluated (line 30). The results are then provided to the optimizer (line 33), which uses them to update the surrogate model (line 36). The process continues until the required number of iterations is completed (line 24). The full set of explored points (line 38) is returned (line 40). This can be used to find the best configurations on the Pareto frontier and select the configuration that best fits the desired requirements.

**Listing 1.** Summarized code of SpikeExplorer.

```
1  from ax.service.ax_client import AxClient
2
3  class SpikExplorer:
4
5      def __init__(self, config: dict):
6
7          self.num_trials     = config.get("num_trials")
8          self.objectives     = config.get("objectives")
9          self.search_param   = config.get("search_param")
10         self.neurons        = config.get("neurons")
11
12     def optimize(self):
13
14         dse_engine = AxClient()
15
16         # Initialize Bayesian optimization
17         dse_engine.create_experiment(
18             parameters=self.search_params,
19             objectives=self.objectives,
20         )
21
22         search_points = []
23
24         for _ in range(self.num_trials):
25
26             # Select initial point in the search space
27             net_config = dse_engine.get_next_trial()
28
29             snn = self.net_generator(net_config)
30             results = self.train_evaluate(snn)
```

```
31
32              # Give the results to the optimizer
33              dse_engine.complete_trial(results)
34
35              # Update the Bayesian surrogate model
36              dse_engine.update()
37
38              search_points.append((net_config, results))
39
40          return search_points
```

Table 1 displays the available optimization parameters, organized into three groups: network architecture (net), neuron model (neuron), and training process (training). Numeric parameters are "discrete" or "continuous" ranges. In the former case, only discrete integer values within the specified range are considered, while in the latter case, a continuous interval of real values is analyzed. Additionally, numeric values can be defined as sets of predefined values to try. For non-numerical parameters, enumerative lists of options are used.

Regarding network architecture, SpikeExplorer offers constraints for optimizing the model. These constraints include the number of layers to use (discrete range), the number of neurons in each layer (set of options), and the network architecture (feed forward or recurrent). As exploring the dimensionality of the network is computationally intensive, selecting the number of neurons per layer from a set allows reduction of the search space by performing a coarser search among a predefined range of layer sizes. Conversely, for a finer search, SpikeExplorer can be left to select any layer size, and a set containing all integer numbers between the desired minimum and maximum can be provided. The final parameter related to the network allows for including recurrent connections within layers. This specification occurs at the network level, configuring the entire network with the specified layer type. Hybrid solutions are not currently considered in the search process. As discussed in Section 4.2, rather than directly specifying whether layers must be recurrent, users can select models that inherently incorporate recurrence.

Nearly all internal parameters can be adjusted at the neuron level after selecting a specific model among the six options listed in Table 1 and elaborated upon in Section 4.2. The reset mechanism can be configured as hard or subtractive (refer to Equation (3)). Optimization of the exponential decay for both the synaptic current and membrane potential can be achieved through the $\alpha$ and $\beta$ parameters ($0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$). In this case, the search can involve continuous values, with users specifying the limits of the search range or selecting from a predefined set of powers of two. The last option aligns with hardware optimization principles, where using powers of two allows replacement of the multiplication involved in exponential decay with a simple bit shift, as demonstrated in [33]. Given that exponential decay generally does not require rapid attenuation, $\alpha$ and $\beta$ typically approach values close to one. Consequently, the search primarily focuses on the upper portion of the interval $[0, 1]$, utilizing the expression outlined in Table 1. Additionally, the firing threshold can be adjusted within a continuous range of values to regulate neuron activity. Finally, users can select the number of time steps involved in computation by specifying a set of values. Similar to the approach for choosing the number of neurons, users can limit the set of sequence lengths, tailoring the set's granularity based on the desired search precision. Alternatively, to grant SpikeExplorer flexibility in selecting from all possible sequence lengths, users can provide a set containing all integer numbers between the desired minimum and maximum.

**Table 1.** Set of specifications that the user can provide.

| | **Parameter** | **Values** | |
|---|---|---|---|
| Net | # layers | Discrete | |
| | # neurons/layer | Set | |
| | Architecture | Feed Forward | Recurrent |
| Neuron | Model | *if*<br>*lif*<br>*syn* | *rif*<br>*rlif*<br>*rsyn* |
| | Reset | Hard | |
| | | Subtractive | |
| | $\alpha$, $\beta$ | Continuous | |
| | | $1 - 2^{-n}$ | |
| | $V_{th}$ | Continuous | |
| | Time-steps | Set | |
| Training | Learning rate | Continuous | |
| | Optimizer | | |
| | Regularizer | | |
| | Surrogate slope | | |
| | Surrogate | Sigmoid, Fast Sigmoid, ATan, Straight Through Estimator, Triangular, SpikeRateEscape, Custom [17] | |

In addition to tuning the network architecture, SpikeExplorer offers optimization options for the training process. This includes fine-tuning parameters such as the learning rate, optimizer settings, such as the Adam [41] parameters $\beta_1$ and $\beta_2$, controlling the decay rates of moving averages of gradients and squared gradients, respectively, and influencing the retention of historical information when updating model parameters—regularization parameters like $\lambda$, affecting the strength of L1 and L2 regularization [42], and modifying the penalty for large weights, and the surrogate function employed in the backward pass, as elaborated in Section 2. In this scenario, SpikeExplorer can select the function itself and adjust its steepness.

Given the many parameters involved, optimization efforts can focus on specific subsets. An illustration of such a targeted search is presented in Section 5.

## 5. Experimental Results

This section demonstrates the capabilities of SpikeExplorer through selected case studies designed to test its internal optimization engine.

### *5.1. Experimental Setup*

The exploration capabilities of SpikeExplorer were evaluated using three distinct datasets, each with varying complexity and characteristics commonly employed for benchmarking SNNs:

1. MNIST [43]: Grayscale images of handwritten digits, converted into sequences of spikes using rate encoding. The corresponding number of inputs is $28 \times 28 = 784$.
2. Spiking Heidelberg Digits (SHD) [44]: Audio recordings of numbers pronounced in English and German, converted to spikes through a faithful emulation of the human cochlea. Recordings were performed with 700 channels, corresponding to the number of inputs of the network.

3. DVS128 [45]: Video recordings of 11 gestures through a Dynamic Vision Sensor (DVS) converting images into spikes. The sensor's resolution is 128 × 128 pixels, accounting for 16,384 inputs.

Two optimization experiments were conducted using the three datasets. In the first experiment, a broad exploration was undertaken, allowing SpikeExplorer the freedom to optimize the training process while seeking optimal neuron models and network architectures. The objective was to minimize area and power consumption while maximizing accuracy. The search parameters provided to SpikeExplorer are detailed in Table 2. The exponential decay rates were set to $\alpha = 0.9$ and $\beta = 0.82$ and were maintained constant throughout the search process. The number of optimization iterations was selected to constrain the optimization time. It was set to 25 for MNIST and DVS128. Conversely, achieving acceptable accuracy with SHD requires more training epochs, so the number of search iterations was capped at 15 to control search duration. The second experiment focused on a more specific optimization goal. Here, the neuron model was fixed initially, and attention shifted to optimizing the total number of neurons within the network.

**Table 2.** Set of experimental parameters provided to SpikeExplorer.

| | **MNIST** | | **SHD** | | **DVS128** | |
|---|---|---|---|---|---|---|
| | **Min** | **Max** | **Min** | **Max** | **Min** | **Max** |
| Learning rate | $10^{-4}$ | $1.2 \times 10^{-4}$ | $10^{-4}$ | $1.2 \times 10^{-4}$ | $10^{-4}$ | $1.2 \times 10^{-4}$ |
| Adam $\beta_{optim}$ | 0.9 | 0.999 | 0.9 | 0.999 | 0.9 | 0.999 |
| # layers | 1 | 3 | 1 | 3 | 1 | 3 |
| Model | lif, syn, rlif, rsyn | | lif, syn, rlif, rsyn | | lif, syn, rlif, rsyn | |
| Reset | subtractive | | subtractive | | subtractive | |
| Time steps | 10, 25, 50 | | 10, 25, 50 | | 10, 25, 50 | |
| # neurons/layers | 200, 100, 50 | | 200, 100, 50 | | 200, 100, 50 | |
| Search iterations | 25 | | 15 | | 25 | |
| Training epochs | 50 | | 100 | | 50 | |

Lastly, hardware synthesis of the optimized architecture identified by SpikeExplorer for the MNIST dataset was conducted to allow for a comparison with SoA FPGA accelerators for SNNs. The dataset is typically used as the reference benchmark to evaluate ML models in general and SNN accelerators specifically. The target hardware platform is a $PYNQ^{TM} - Z2$ board, from $TUL^{®}$ hosting a $Xilinx^{®}$ $Zynq - 7000$ $XC7Z020 - 1CLG400C$ system on chip (SoC). This features the $XC7Z020$ FPGA, and a Dual $ARM^{®}$ $Cortex^{TM} - A9$ $MPCore^{TM}$. The FPGA can be programmed with the free version of the $Xilinx^{®}$ Vivado suite, making the results strongly reproducible.

### *5.2. Global Exploration*

Figure 4 summarizes the performance of SpikeExplorer when optimizing the network architecture and parameters for the three selected datasets. The figure demonstrates a strong correlation between power consumption (the first row in Figure 4) and area (the second row in Figure 4). While this behavior is expected, it is noteworthy because previous publications predominantly emphasized the correlation between power consumption and spiking activity [33]. For the MNIST dataset (refer to Figure 4a,d, nonrecurrent models emerge as the preferred choice. This preference is evident from the Pareto frontier, where virtually all top-performing models are *lif* and *syn* without recurrence. Notably, the highest accuracy is achieved with a first-order LIF model, devoid of any feedback connection (refer to Table 3). This is consistent with expectations since simple architectures without explicit recurrent connections should be enough, given the static nature of MNIST data transformed

into spike sequences via rate coding. In this scenario, crucial information is not embedded in the temporal dimension but encoded in the average spike sequence rate. Consistently with what is expected, SpikeExplorer converges towards these more straightforward solutions. Conversely, in the case of SHD and DVS128, acquired through biologically inspired sensors and containing substantial information in spike timing, SpikeExplorer generally leans towards recurrent structures such as *rlif* and *rsyn*, along with higher-order models (*syn*). Specifically, for SHD, a recurrent structure comprising *rlif* neurons emerges as the favored solution. At the same time, the search tends to diversify more toward both *rsyn* and *rlif*, occasionally incorporating *syn* instances for the DVS128. It is noteworthy to observe how SpikeExplorer can discover superior architectures in terms of accuracy by utilizing the same neuron model and comparable numbers of neurons while playing on other parameters, allowing us to keep the power consumption unchanged while better tuning them on the target task. This is visible when looking at the left section of the Pareto frontier across all three datasets.

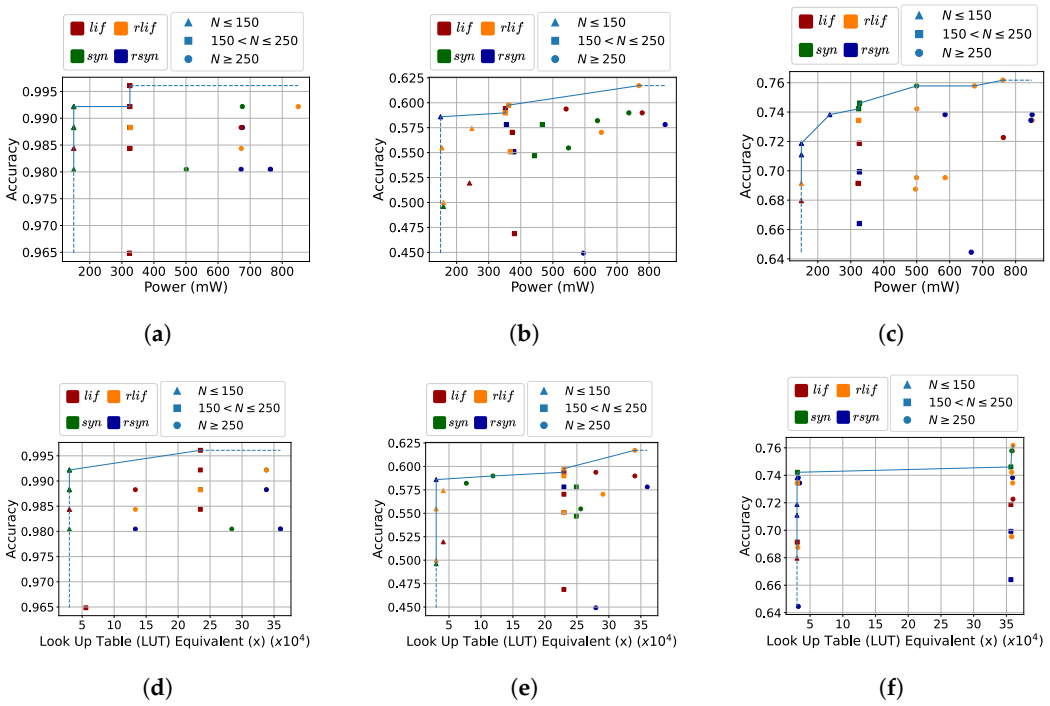



**Figure 4.** Pareto frontiers of the global exploration on the three benchmark datasets targeting power, area, and accuracy optimization. (**a**) MNIST complete power; (**b**) SHD complete power; (**c**) DVS complete power; (**d**) MNIST complete area; (**e**) SHD complete area; (**f**) DVS complete area.

In summary, Tables 3–5 showcase the top-1 accuracy optimized SNN architecture, parameters, and performance identified by SpikeExplorer for the three benchmarks, categorized by neuron model. The optimization is performed according to the setup summarized in Table 2.

**Table 3.** Best architectures with the four neuron models on the MNIST.

| Model | Arch. | TS | Acc. | Power (mW) |
|---|---|---|---|---|
| LIF | 200-10 | 10 | 99.61% | 310 |
| RLIF | 200-100-200-10 | 10 | 99.22% | 860 |
| SYN | 200-200-10 | 25 | 99.22% | 680 |
| RSYN | 100-10 | 25 | 99.22% | 140 |

**Table 4.** Best architectures with the four neuron models on the SHD.

| Model | Arch. | TS | Acc. | Power (mW) |
|---|---|---|---|---|
| LIF | 200-20 | 50 | 59.41% | 360 |
| RLIF | 200-200-20 | 50 | 61.70% | 760 |
| SYN | 100-100-200-20 | 10 | 58.98% | 720 |
| RSYN | 100-20 | 50 | 58.59% | 140 |

**Table 5.** Best architectures with the four neuron models on the DVS.

| Model | Arch. | TS | Acc. | Power (mW) |
|---|---|---|---|---|
| LIF | 200-200-50-11 | 50 | 72.27% | 500 |
| RLIF | 200-200-50-11 | 25 | 76.17% | 760 |
| SYN | 200-100-11 | 50 | 75.78% | 500 |
| RSYN | 100-200-50-10 | 50 | 73.83% | 590 |

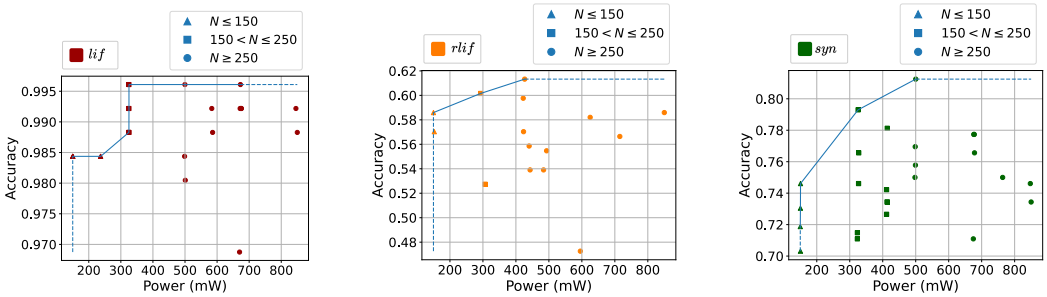
To showcase the capability of SpikeExplorer across different use cases, this study limited the maximum number of time steps to 50 to mitigate training time. However, upon reviewing the accuracy achieved by the optimized models, it seems reasonable to assume that these datasets may benefit from longer sequences. For instance, ref. [33] reports a 75% accuracy for SHD with a 200-20 network using *rsyn* neurons and 100 time steps. Conversely, models tailored for MNIST can achieve nearly SoA accuracies with minimal time steps. Regarding architectures, the search for DVS128 tends toward larger structures, which correspondingly increases power consumption.

In terms of computing time, the exploration took approximately 5 h for both MNIST and DVS, and approximately 16 h for SHD, conducted on an AMD Ryzen 9 7950X 16-Core Processor and an Nvidia RTX400 GPU. It is worth noting that the primary time consumption arises from training the network, which is more time-intensive for recurrent models than nonrecurrent models due to the inability to accelerate the explicit time dependence through GPU.

### *5.3. Fixed Neuron Models and Network Size*

After showcasing the overall optimization capabilities of SpikeExplorer, additional experiments were performed to highlight its behavior in constrained optimization problems.

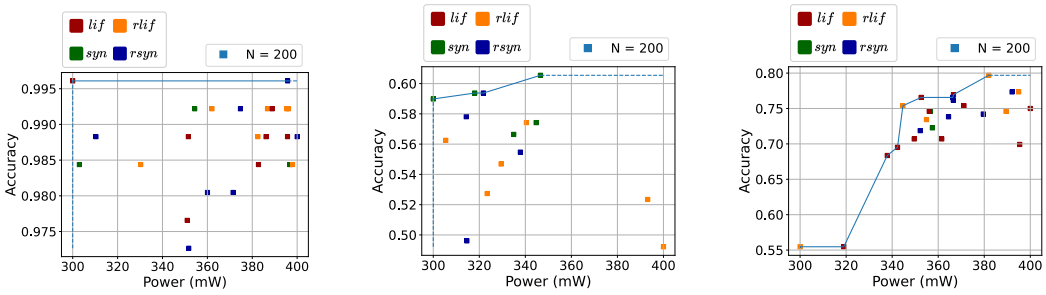
Figure 5 showcases the capability of SpikeExplorer to optimize the network with a predefined neuron model, solely using the network architecture and parameters. In this case, the Pareto frontiers are dominated by small architectures ($N \leq 250$). It is interesting to observe that the optimizer is very effective when selecting the network architecture: for example, for the MNIST, an architecture with 200-10 neurons (square on the top left of Pareto frontier) can obtain the same accuracy of bigger solutions (circles on the top right of the Pareto frontier), reducing the power requirements by factors of 1.5 and more than 2, respectively. Results reported in Figure 5 also highlight the capability of SpikeExplorer in supporting designers in finding the suitable trade-off between different metrics. For example, the Pareto frontier in Figure 5b shows that the power can be reduced almost three times by accepting an accuracy loss of 3%. Interestingly, the accuracy on the DVS128 is pushed up to the best value of 81.6%, improving by around 5% concerning a more agnostic search, indicating that a more specialized search can reach even better results.




(**a**) MNIST best model power (**b**) SHD best model power (**c**) DVS best model power

**Figure 5.** Pareto frontiers of the exploration with top-accuracy neuron model for each benchmark.

Finally, Figure 6 shows the behavior of SpikeExplorer when constraining the total number of neurons to 200 to study how different models behave. Interestingly, this produces different observations compared to results reported in Section 5.2. The optimization for SHD now privileges the *syn* model, either with or without recurrent connections, while the *lif* model dominates the Pareto frontier in the DVS128 case. Again, the top-1 accuracy is increased, even if it is less than in the search with a fixed neuron model, reaching around 80%. This again supports the utility of a tool like SpikeExplorer when exploring different design opportunities.




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

**Figure 6.** Pareto frontiers of the exploration with the number of neurons constrained to 200 for each benchmark. (**a**) MNIST fixed number of neurons power; (**b**) SHD fixed number of neurons power; (**c**) DVS fixed number of neurons power.

### *5.4. Synthesis and Comparison with State of Art*

As discussed in Section 4, the power and area values provided by SpikeExplorer are estimations to guide the DSE process and do not represent the actual values of the final FPGA implementation of the respective model. To obtain actual values and compare the performance of the models optimized by SpikeExplorer with SoA SNN accelerators designed for FPGAs, a synthesis of an optimized architecture was conducted using the Spiker+ framework to generate the VHDL description [33]. This process generated the hardware implementation of a 128-10 architecture optimized with SpikeExplorer for the MNIST dataset. This architecture is compared with other accelerators in Table 6.

An important observation is that the same architecture, with the same neuron model used in [33], is considered for a direct comparison. All other parameters are optimized following the approach outlined in Sections 5.2 and 5.3. In this scenario, SpikeExplorer optimizes the model by reducing the time steps from 100 to 16, decreasing the overall latency by more than six times, from 780 µs to 120 µs. Simultaneously, the optimized training increases the accuracy by almost 3%, reaching 95.8%, thereby establishing the new optimized model as the best one among those considered, both in terms of power consumption and latency, while also positioning it close to the best-performing model in

terms of accuracy [29]. Thus, SpikeExplorer demonstrates its capability to enhance the design of FPGA accelerators for SNNs, simplifying the selection of the optimal architecture and effectively tailoring it to the desired application. It must be noted that SpikeExplorer can optimize a target accelerator, starting from an existing set of hardware blocks. If the goal is to optimize latency and power further, the tool requires a more efficient neuron implementation tailored explicitly for low-power or high-performance applications.

**Table 6.** Comparison of SpikeExplorer to state-of-the-art FPGA accelerators for SNNs.

| Design | Han et al. [29] | Gupta et al. [30] | Li et al. [31] | Spiker [32] | Spiker+ [33] | This Work |
|---|---|---|---|---|---|---|
| Year | 2020 | 2020 | 2021 | 2022 | 2024 | |
| $f_{clk}$ [MHz] | 200 | 100 | 100 | 100 | | |
| Neuron bw | 16 | 24 | 16 | 16 | 6 | |
| Weights bw | 16 | 24 | 16 | 16 | 4 | |
| Update | Event | Event | Hybrid | Clock | | |
| Model | LIF | LIF [46] | LIF | LIF | | |
| FPGA | XC7Z045 | XC6VLX240T | XC7VX485 | XC7Z020 | | |
| Avail. BRAM | 545 | 416 | 2060 | 140 | | |
| Used BRAM | 40.5 | 162 | N/R | 45 | 18 | |
| Avail. DSP | 900 | 768 | 2800 | 220 | | |
| Used DSP | 0 | 64 | N/R | 0 | | |
| Avail. logic cells | 655,800 | 452,160 | 485,760 | 159,600 | | |
| Used logic cells | 12,690 | 79,468 | N/R | 55,998 | 7612 | |
| Arch | 1024-1024-10 | 784-16 | 200-100-10 | 400 | 128-10 | |
| #syn | 1,861,632 | 12,544 | 177,800 | 313,600 | 101,632 | |
| $T_{lat}$/img [ms] | 6.21 | 0.50 | 3.15 | 0.22 | 0.78 | 0.12 |
| Power [W] | 0.477 | N/R | 1.6 | 59.09 | 0.18 | |
| E/img [mJ] | 2.96 | N/R | 5.04 | 13 | 0.14 | 0.02 |
| E/syn [nJ] | 1.59 | N/R | 28 | 41 | 1.37 | 0.22 |
| Accuracy | 97.06% | N/R | 92.93% | 73.96% | 93.85% | 95.8% |

## 6. Conclusions and Future Work

This paper introduced SpikeExplorer, a tool tailored for hardware-centric ADSE in SNNs. Specifically designed for crafting and fine-tuning specialized hardware accelerators intended for deployment on FPGA, this tool showcases the effectiveness of Bayesian optimization within the context of SNNs. It enables an easy and flexible multiobjective search, considering model accuracy and critical hardware-specific metrics such as power consumption, area utilization, and latency. The design of SpikeExplorer builds upon three open-source projects: snnTorch, AX, and Spiker+. Being open-source, SpikeExplorer offers a robust solution for optimizing SNNs.

The capabilities of SpikeExplorer were evaluated across three distinct tasks: static image recognition using the MNIST dataset, a prevalent benchmark in ML; speech recognition on the SHD dataset; and gesture recognition on the DVS128 dataset. In the MNIST scenario, the tool achieved outstanding performance, surpassing existing solutions in terms of latency by classifying images in approximately 120 µs while consuming minimal power (180 mW) and achieving high accuracy (95.8%). On the SHD task, it encountered challenges, achieving a top-1 accuracy of approximately 62%, possibly due to the limited number of time steps used for spike sequences during optimization. Regarding the DVS128 dataset, SpikeExplorer delivered promising results, achieving 81.6% top-1 accuracy. Notably, the high dimensionality of the inputs of this dataset, with 128 × 128 event-based channels, made the use of FC networks suboptimal and fully parallel processing infeasible. Nevertheless, this dataset served as a valuable case study for evaluating the optimization tool with a complex dataset.

Despite the promising experimental results, additional testing with more complex case studies will be conducted to identify and solve cold boot and scalability issues that may affect Bayesian optimization. Future work also involves expanding the framework's scope to encompass different architectures such as Convolutional Spiking Neural Network (CSNN) and generalizing the tool to accommodate diverse computing paradigms like event-driven processors. Despite not being explicitly tailored for such hardware accelerators,

SpikeExplorer exhibits considerable flexibility, supporting custom neuron models and configurable metric assessments during optimization. This lays a solid foundation for automating the optimization of SNN co-processors, thereby facilitating the adoption of neuromorphic solutions in resource-constrained edge applications.

**Author Contributions:** D.P. and A.C. worked on conceptualization, methodology, software development, validation, and data curation. A.C., A.S. and S.D.C. contributed to paper writing and supervision. S.D.C. and A.S. contributed to project administration and funding acquisition. All authors have read and agreed to the published version of the manuscript.

**Funding:** This paper has received funding from: The NEUROPULS project in the European Union's Horizon Europe research and innovation programme under grant agreement No. 101070238; The APROPOS project in the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 956090; The project "National Center for HPC, Big Data and Quantum Computing", CN00000013 (Bando M42C–Investimento 1.4–Avviso Centri Nazionali"–D.D. n. 3138 of 16 December 2021, funded with MUR Decree n. 1031 of 17 June 2022). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** To encourage research in this field, SpikeExplorer is released open-source on GitHub at, accessed on 1 April 2024 https://github.com/smilies-polito/SpikeExplorer.

**Conflicts of Interest:** The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

## References

1. Kasabov, N.K. *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*; Springer Series on Bio- and Neurosystems; Springer: Berlin/Heidelberg, Germany, 2019; Volume 7. [CrossRef]
2. Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Netw.* **1997**, *10*, 1659–1671. [CrossRef]
3. Narayanan, S.; Taht, K.; Balasubramonian, R.; Giacomin, E.; Gaillardon, P.E. SpinalFlow: An Architecture and Dataflow Tailored for Spiking Neural Networks. In Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 30 May–3 June 2020; pp. 349–362. [CrossRef]
4. Basu, A.; Frenkel, C.; Deng, L.; Zhang, X. Spiking Neural Network Integrated Circuits: A Review of Trends and Future Directions. *arXiv* **2022**, arXiv:2203.07006.
5. Isik, M. A Survey of Spiking Neural Network Accelerator on FPGA. *arXiv* **2023**, arXiv:2307.03910.
6. Musisi-Nkambwe, M.; Afshari, S.; Barnaby, H.; Kozicki, M.; Esqueda, I.S. The viability of analog-based accelerators for neuromorphic computing: A survey. *Neuromorphic Comput. Eng.* **2021**, *1*, 012001. [CrossRef]
7. Wang, T.T.; Chu, S.C.; Hu, C.C.; Jia, H.D.; Pan, J.S. Efficient Network Architecture Search Using Hybrid Optimizer. *Entropy* **2022**, *24*, 656. [CrossRef] [PubMed]
8. Ghaffari, A.; Savaria, Y. CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration. *Electronics* **2020**, *9*, 2200. [CrossRef]
9. Czako, Z.; Sebestyen, G.; Hangan, A. AutomaticAI–A hybrid approach for automatic artificial intelligence algorithm selection and hyperparameter tuning. *Expert Syst. Appl.* **2021**, *182*, 115225. [CrossRef]
10. Balaji, A.; Song, S.; Titirsha, T.; Das, A.; Krichmar, J.; Dutt, N.; Shackleford, J.; Kandasamy, N.; Catthoor, F. NeuroXplorer 1.0: An Extensible Framework for Architectural Exploration with Spiking Neural Networks. In Proceedings of the International Conference on Neuromorphic Systems 2021, ICONS 2021, Knoxville, TN, USA, 27–29 July 2021; pp. 1–9. [CrossRef]
11. Abderrahmane, N.; Lemaire, E.; Miramond, B. Design Space Exploration of Hardware Spiking Neurons for Embedded Artificial Intelligence. *Neural Netw.* **2020**, *121*, 366–386. [CrossRef] [PubMed]
12. Xue, J.; Xie, L.; Chen, F.; Wu, L.; Tian, Q.; Zhou, Y.; Ying, R.; Liu, P. EdgeMap: An Optimized Mapping Toolchain for Spiking Neural Network in Edge Computing. *Sensors* **2023**, *23*, 6548. [CrossRef]
13. Samsi, S.; Zhao, D.; McDonald, J.; Li, B.; Michaleas, A.; Jones, M.; Bergeron, W.; Kepner, J.; Tiwari, D.; Gadepally, V. From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference. *arXiv* **2023**, arXiv:2310.03003.
14. Hodgkin, A.L.; Huxley, A.F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* **1952**, *117*, 500–544. [CrossRef] [PubMed]

15. Izhikevich, E. Simple model of spiking neurons. *IEEE Trans. Neural Netw.* **2003**, *14*, 1569–1572. [CrossRef] [PubMed]
16. Brunel, N.; van Rossum, M.C.W. Quantitative investigations of electrical nerve excitation treated as polarization. *Biol. Cybern.* **2007**, *97*, 341–349. [CrossRef] [PubMed]
17. Eshraghian, J.K.; Ward, M.; Neftci, E.; Wang, X.; Lenz, G.; Dwivedi, G.; Bennamoun, M.; Jeong, D.S.; Lu, W.D. Training Spiking Neural Networks Using Lessons From Deep Learning. *arXiv* **2021**, arXiv:2109.12894.
18. Markram, H.; Gerstner, W.; Sjöström, P.J. Spike-Timing-Dependent Plasticity: A Comprehensive Overview. *Front. Synaptic Neurosci.* **2012**, *4*, 2. [CrossRef] [PubMed]
19. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
20. Neftci, E.O.; Mostafa, H.; Zenke, F. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks. *IEEE Signal Process. Mag.* **2019**, *36*, 51–63. [CrossRef]
21. Marti, K. *Optimization under Stochastic Uncertainty: Methods, Control and Random Search Methods*; International Series in Operations Research & Management Science; Springer International Publishing: Cham, Switzerland, 2020; Volume 296. [CrossRef]
22. Ferrandi, F.; Lanzi, P.L.; Loiacono, D.; Pilato, C.; Sciuto, D. A Multi-objective Genetic Algorithm for Design Space Exploration in High-Level Synthesis. In Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 7–9 April 2008; pp. 417–422. ISSN 2159-3477. [CrossRef]
23. Savino, A.; Vallero, A.; Di Carlo, S. ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems. *IEEE Trans. Comput.* **2018**, *67*, 1462–1477. [CrossRef]
24. Saeedi, S.; Savino, A.; Di Carlo, S. Design Space Exploration of Approximate Computing Techniques with a Reinforcement Learning Approach. In Proceedings of the 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Porto, Portugal, 27–30 June 2023; pp. 167–170. ISSN 2325-6664. [CrossRef]
25. Reagen, B.; Hernández-Lobato, J.M.; Adolf, R.; Gelbart, M.; Whatmough, P.; Wei, G.Y.; Brooks, D. A case for efficient accelerator design space exploration via Bayesian optimization. In Proceedings of the 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Taipei, Taiwan, 24–26 July 2017; pp. 1–6. [CrossRef]
26. March, J.G. Exploration and Exploitation in Organizational Learning. *Organ. Sci.* **1991**, *2*, 71–87. [CrossRef]
27. Candelieri, A. A Gentle Introduction to Bayesian Optimization. In Proceedings of the 2021 Winter Simulation Conference (WSC), Phoenix, AZ, USA, 12–15 December 2021; pp. 1–16. [CrossRef]
28. Bouvier, M.; Valentian, A.; Mesquida, T.; Rummens, F.; Reyboz, M.; Vianello, E.; Beigne, E. Spiking Neural Networks Hardware Implementations and Challenges: A Survey. *J. Emerg. Technol. Comput. Syst.* **2019**, *15*, 1–35. [CrossRef]
29. Han, J.; Li, Z.; Zheng, W.; Zhang, Y. Hardware implementation of spiking neural networks on FPGA. *Tsinghua Sci. Technol.* **2020**, *25*, 479–486. [CrossRef]
30. Gupta, S.; Vyas, A.; Trivedi, G. FPGA Implementation of Simplified Spiking Neural Network. In Proceedings of the 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, UK, 23–25 November 2020; pp. 1–4. [CrossRef]
31. Li, S.; Zhang, Z.; Mao, R.; Xiao, J.; Chang, L.; Zhou, J. A Fast and Energy-Efficient SNN Processor With Adaptive Clock/Event-Driven Computation Scheme and Online Learning. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 1543–1552.[CrossRef]
32. Carpegna, A.; Savino, A.; Di Carlo, S. Spiker: An FPGA-optimized Hardware accelerator for Spiking Neural Networks. In Proceedings of the 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Nicosia, Cyprus, 4–6 July 2022; pp. 14–19. ISSN 2159-3477. [CrossRef]
33. Carpegna, A.; Savino, A.; Di Carlo, S. Spiker+: A framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge. *arXiv* **2024**, arXiv:2401.01141.
34. Gerlinghoff, D.; Wang, Z.; Gu, X.; Goh, R.S.M.; Luo, T. E3NE: An End-to-End Framework for Accelerating Spiking Neural Networks With Emerging Neural Encoding on FPGAs. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 3207–3219. [CrossRef]
35. Kim, Y.; Li, Y.; Park, H.; Venkatesha, Y.; Panda, P. Neural Architecture Search for Spiking Neural Networks. In Proceedings of the Computer Vision—ECCV 2022, Tel Aviv, Israel, 23–27 October 2022; Lecture Notes in Computer Science; Avidan, S., Brostow, G., Cissé, M., Farinella, G.M., Hassner, T., Eds.; Springer: Cham, Switzerland, 2022; pp. 36–56. [CrossRef]
36. Moradi, S.; Qiao, N.; Stefanini, F.; Indiveri, G. A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs). *IEEE Trans. Biomed. Circuits Syst.* **2018**, *12*, 106–122. [CrossRef] [PubMed]
37. Putra, R.V.W.; Shafique, M. Q-SpiNN: A Framework for Quantizing Spiking Neural Networks. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–8. ISBN 9781665439008. [CrossRef]
38. Li, C.; Ma, L.; Furber, S. Quantization Framework for Fast Spiking Neural Networks. *Front. Neurosci.* **2022**, *16*, 918793. [CrossRef] [PubMed]
39. Castagnetti, A.; Pegatoquet, A.; Miramond, B. Trainable quantization for Speedy Spiking Neural Networks. *Front. Neurosci.* **2023**, *17*, 1154241. [CrossRef]
40. Meta. Ax · Adaptive Experimentation Platform—ax.dev. Available online: https://ax.dev (accessed on 3 April 2024).
41. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.

42. Ng, A.Y. Feature selection, L1 vs. L2 regularization, and rotational invariance. In Proceedings of the Twenty-First International Conference on Machine Learning, ICML '04, Banff, AB, Canada, 4–8 July 2004; p. 78. [CrossRef]
43. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
44. Cramer, B.; Stradmann, Y.; Schemmel, J.; Zenke, F. The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 2744–2757. [CrossRef]
45. Amir, A.; Taba, B.; Berg, D.; Melano, T.; McKinstry, J.; Di Nolfo, C.; Nayak, T.; Andreopoulos, A.; Garreau, G.; Mendoza, M.; et al. A Low Power, Fully Event-Based Gesture Recognition System. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 7388–7397. [CrossRef]
46. Iakymchuk, T.; Rosado-Muñoz, A.; Guerrero-Martínez, J.F.; Bataller-Mompeán, M.; Francés-Víllora, J.V. Simplified spiking neural network architecture and STDP learning algorithm applied to image classification. *EURASIP J. Image Video Process.* **2015**, *2015*, 4. [CrossRef]