Software-Level Sparsity Optimization for Low-Power Spiking Neural Networks

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)
angenommene

DISSERTATION

von

M.Sc. George-Alexandru Vasilache

geboren in: Iasi - Rumänien

Tag der mündlichen Prüfung: 20.11.2025

Hauptreferent: Prof. Dr.-Ing. Jürgen Becker
Korreferentin: Prof. Dr. Yulia Sandamirskaya

Abstract

The computational demands of deep learning models present a challenge for their deployment in energy-constrained environments. Neuromorphic computing, which draws inspiration from the brain's computational principles, offers an alternative through the use of Spiking Neural Networks (SNNs) that communicate via sparse, event-driven signals. This dissertation develops software-level strategies for managing the energy consumption of SNNs, guided by an estimation framework that approximates energy usage as a product of hardware-specific energy per synaptic operation (E_{synop}), average neuron activations (N_{spikes}), and the number of synaptic connections ($N_{\text{connections}}$). This work presents methods to optimize the latter two terms: activation sparsity (N_{spikes}) and connection sparsity ($N_{\text{connections}}$).

To improve activation sparsity, two contributions are made. First, an open-source framework for spike encoding is introduced, which converts continuous data into sparse spike trains, reducing data density by up to 5.6× while preserving information. Second, this work integrates spiking neurons into established deep learning architectures, including Convolutional Neural Networks (CNNs) and recurrent layers. In applications covering sleep analysis, neural decoding, and predictive maintenance, these hybrid models achieved activation sparsities of 84–94%, potentially reducing the number of spike-driven computations by a factor of 6× to 16×. When optimized for performance, the same models achieved state-of-the-art results in two of the three tasks.

To improve connection sparsity, this work introduces methods inspired by the spatial organization of biological neural systems as an alternative to post-hoc pruning alone. One approach embeds neurons in Euclidean space, allowing the network's topology to be optimized via gradient-based methods, which reduced connections by 80% with no significant loss in accuracy. Another approach uses gradient-free evolutionary algorithms to co-evolve the topology and parameters of SNNs for control tasks in

robotics. This method produced controllers with up to 99.94% fewer non-zero connections than baseline models, a reduction of over 1600×, while maintaining competitive performance.

The proposed methods are estimated to reduce energy consumption by one to three orders of magnitude, and when paired with neuromorphic hardware on a specific use case, total energy consumption is estimated to be up to 5000× lower than x86 and 500× lower than ARM processors. All in all, this work advances the practical deployment of SNNs on resource-constrained hardware through software-level strategies for inducing sparsity.

Zusammenfassung

Die Rechenanforderungen von Deep-Learning-Modellen erschweren ihren Einsatz in energiebegrenzten Umgebungen. Die neuromorphe Datenverarbeitung, die sich an den rechnerischen Prinzipien des Gehirns orientiert, bietet mit Spiking Neural Networks (SNNs), die über spärliche, ereignisgetriebene Signale kommunizieren, eine Alternative. Diese Dissertation entwickelt softwareseitige Strategien zur Steuerung des Energieverbrauchs von SNNs, geleitet von einem Abschätzungsrahmen, der den Energiebedarf als Produkt aus hardwarespezifischer Energie pro synaptischer Operation (E_{synop}), mittlerer Neuronaktivität (N_{spikes}) und der Anzahl synaptischer Verbindungen ($N_{\text{connections}}$) approximiert. Es werden Verfahren vorgestellt, die die beiden letzteren Terme optimieren: Sparsity der Aktivierungen (N_{spikes}) und Sparsity der Verbindungen ($N_{\text{connections}}$).

Zur Erhöhung der Sparsity der Aktivierungen werden zwei Beiträge geleistet. Erstens wird ein Framework für Spike-Codierung vorgestellt, das kontinuierliche Daten in spärliche Spike-Züge überführt und die Datendichte um bis zu 5,6× reduziert, bei Erhalt der Information. Zweitens integriert diese Arbeit spikende Neuronen in etablierte Deep-Learning-Architekturen, darunter Convolutional Neural Networks (CNNs) und rekurrente Schichten. In Anwendungen wie Schlafanalyse, neuronale Dekodierung und Predictive Maintenance erreichten diese hybriden Modelle eine Aktivierungssparsity von 84–94% und verringerten potenziell die Anzahl der spikegetriebenen Berechnungen um den Faktor 6× bis 16×. Bei Leistungsoptimierung erzielten dieselben Modelle in zwei der drei Aufgaben Ergebnisse auf dem neuesten Stand der Technik.

Zur Verbesserung der Sparsity der Verbindungen werden in dieser Arbeit Methoden vorgestellt, die von der räumlichen Organisation biologischer neuronaler Systeme inspiriert sind und eine Alternative zum nachträglichen Pruning allein bieten. Ein Ansatz bettet Neuronen in den euklidischen Raum ein und ermöglicht die Optimierung der Netzwerktopologie mittels

gradientenbasierter Methoden; damit ließen sich Verbindungen um 80% reduzieren, ohne nennenswerte Genauigkeitseinbußen. Ein weiterer Ansatz verwendet gradientenfreie evolutionäre Algorithmen, um Topologie und Parameter von SNNs für Regelungsaufgaben in der Robotik gemeinsam zu evolvieren. Damit wurden Regler mit bis zu 99,94% weniger nicht-null-Verbindungen als Baseline-Modellen erzeugt, eine Reduktion um über 1600×, bei wettbewerbsfähiger Leistung.

Die vorgeschlagenen Verfahren werden auf eine Reduktion des Energieverbrauchs um ein bis drei Größenordnungen geschätzt; in Kombination mit neuromorpher Hardware in einem konkreten Anwendungsfall wird der gesamte Energieverbrauch auf bis zu 5000× geringer als bei x86- und 500× geringer als bei ARM-Prozessoren abgeschätzt. Insgesamt leistet diese Arbeit einen Beitrag zur praktischen Einsatzfähigkeit von SNNs auf ressourcenbeschränkter Hardware durch softwareseitige Strategien zur Induktion von Sparsity.

Contents

1. Introduction

1.1. Motivation

The proliferation of artificial intelligence and deep learning has led to increasingly powerful models, but this progress has come at the cost of significant energy consumption. The computational demands of state-of-the-art Artificial Neural Networks (ANNs) often render them unsuitable for deployment in resource-constrained environments, such as battery-powered edge devices, mobile robotics, and implantable biomedical systems. This limitation has motivated a shift toward bio-inspired computing paradigms that promise greater efficiency.

Neuromorphic computing, inspired by the structure and function of the brain, provides an alternative approach to conventional artificial intelligence. Central to this paradigm are SNNs, which differ from traditional neural networks by communicating through discrete, sparse pulses known as "spikes" [58]. This event-driven mechanism enables computation to occur only when required, which supports energy efficiency. The development of neuromorphic hardware, specifically designed to process sparse, spike-based signals, has further increased the relevance of this approach by enabling lower energy consumption compared to conventional platforms. As a result, the energy efficiency of SNNs is most apparent when they are deployed on hardware tailored to their operational characteristics. Achieving these benefits, however, necessitates both conceptual and practical changes from traditional deep learning methodologies.

The central challenge lies in effectively harnessing the sparsity that makes these networks efficient. This dissertation is guided by the principle that part of the energy footprint of an SNN can be managed at the software level. This is formalized in the energy estimation framework introduced in Chapter 3, which decomposes the problem into three components: the

hardware-specific energy per synaptic operation (E_{synop}), the average number of neuron activations (N_{spikes}), and the total number of synaptic connections ($N_{\text{connections}}$). As this work focuses on software-level improvements, the research is concentrated on developing strategies to optimize the latter two terms: activation sparsity and connection sparsity.

1.2. Research Objectives and Contributions

The primary objective of this dissertation is to develop, implement, and evaluate a set of software-level strategies for inducing activation and connection sparsity in SNNs to improve their energy efficiency. The main contributions are organized around these two central themes.

To improve activation sparsity, this work makes two primary contributions. First, it addresses the need for sparse data representations at the network's input by developing a flexible, open-source spike encoding framework [VSS+25]. This tool enables the conversion of continuous, real-world data into sparse spike trains suitable for SNN processing, while providing mechanisms for optimizing the trade-off between information preservation and sparsity. Second, this research explores the integration of spiking neurons into established ANN architectures. By replacing conventional activation functions with spiking dynamics in Convolutional Neural Networks (CNNs) [BVH+24] and recurrent layers [VKKB24, KVKB25, VNK+25], this work demonstrates that the benefits of event-driven computation can be leveraged within familiar deep learning frameworks.

To improve connection sparsity, this dissertation introduces novel methods inspired by the spatial organization of biological neural systems, moving beyond traditional post-hoc pruning techniques. The first approach involves embedding neurons in Euclidean space and making their connection weights a differentiable function of their learnable spatial coordinates, enabling the network's topology to be optimized alongside its weights using gradient-based methods [EBV+25]. The second approach uses gradient-free evolutionary algorithms to optimize SNNs whose neurons are situated in fixed spatial grids [VSSB25]. This allows for the co-evolution of network topology and complex neuron parameters, resulting in inherently sparse and efficient architectures from the outset.

1.3. Dissertation Outline

Chapter 2: Fundamentals provides the necessary background on SNNs, including common neuron models, spike encoding techniques, training paradigms, and the neuromorphic hardware designed to execute them efficiently.

Chapter 3: Concept details the energy estimation framework that serves as the guiding principle for this research, providing the conceptual basis for targeting activation and connection sparsity as a means to reduce energy consumption.

Chapter 4: Related Work reviews the existing literature on activation and connection sparsity, identifies the specific research gaps that this dissertation addresses, and defines the benchmark tasks used to evaluate the proposed methods.

Chapter 5: Methods presents the specific software-level methods developed in this work, organized according to the two primary objectives of improving activation sparsity and connection sparsity.

Chapter 6: Results presents the empirical results from applying the proposed methods to the benchmark tasks, providing a quantitative evaluation of their performance and efficiency.

Chapter 7: Discussion synthesizes and interprets the key findings of this research, discusses the limitations of the work, and outlines directions for future research.

Chapter 8: Conclusion summarizes the main contributions of the dissertation and offers concluding remarks.

2. Fundamentals

This chapter provides the foundational knowledge required to understand the concepts and contributions presented in this dissertation. It begins by introducing the core principles of SNNs, contrasting them with conventional ANNs and detailing the dynamics of common spiking neuron models. Subsequently, it addresses the topic of spike encoding, reviewing the methods used to translate continuous data into an event-based format. The chapter then surveys the primary paradigms for training SNNs, from backpropagation-based techniques to local plasticity rules and evolutionary algorithms. Finally, it provides an overview of neuromorphic hardware, the specialized computing platforms designed to exploit the inherent efficiency of SNNs.

2.1. Spiking Neural Networks (SNNs)

SNNs, often referred to as the third generation of neural networks, represent a paradigm shift from conventional ANNs by more closely mimicking the computational principles of the brain. While ANNs operate on continuous-valued activations, SNNs communicate through discrete, sparse pulses, or "spikes", similar to biological neurons. This event-driven nature makes SNNs inherently efficient, as computation is performed only when a spike occurs. This property, when combined with specialized neuromorphic hardware, enables significant reductions in energy consumption and latency compared to traditional Deep Learning (DL) models [58].

The fundamental difference between ANNs and SNNs lies in their respective neuron models. In an ANN, a neuron computes a weighted sum of its inputs and applies a non-linear activation function to produce a continuous output value. In contrast, a spiking neuron, governed by a set of differential equations, integrates its inputs over time, dynamically changing its internal

state, which is typically represented by its membrane potential. The inspiration for these models comes from biological neurons, which consist of dendrites that receive signals, a soma that integrates them, and an axon that transmits the resulting spike to other neurons via synapses.

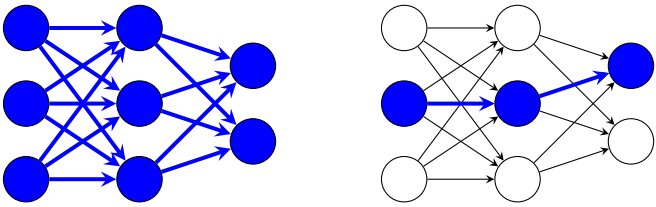


Figure 2.1.: Comparison between an ANN and an SNN. In the ANN (left), all neurons are active and contribute to the computation at every step. In the SNN (right), only a sparse subset of neurons are active at any given time (highlighted in blue), leading to event-driven computation.

A key characteristic of SNNs is their sparse activation, as shown in Figure 2.1. Unlike in dense ANNs where all neurons compute a value at every forward pass, in SNNs only the neurons that receive spikes and subsequently reach their firing threshold are active. This spatio-temporal sparsity is a primary source of their computational efficiency. On hardware designed to exploit this property, inactive neurons consume little to no power, which can reduce the overall energy cost of computation. This makes SNNs a technology for applications in resource-constrained environments, such as edge devices and mobile robotics.

Neuron Models

The behavior of an SNN is fundamentally determined by the dynamics of its constituent neurons. A wide spectrum of spiking neuron models exists, ranging from computationally simple abstractions to biologically detailed representations. The choice of model involves a trade-off between computational complexity and biological plausibility.

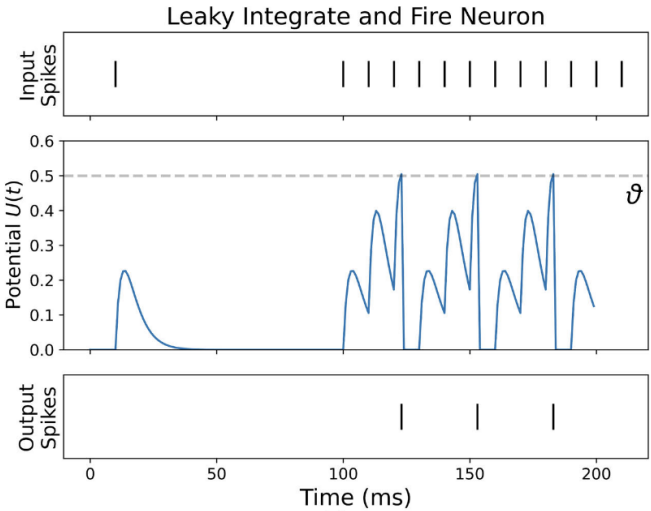


Figure 2.2.: An illustration of the voltage dynamics in a Leaky Integrate-and-Fire (LIF) neuron. As incoming spikes arrive, the membrane potential rises. In the absence of input, the potential decays back to its resting state. When the potential surpasses a threshold, the neuron fires a spike and its potential is reset. Adapted from [Erb25].

Leaky Integrate-and-Fire (LIF) Model: The most widely used spiking neuron model, due to its computational simplicity, is the LIF model. The LIF neuron's membrane potential, illustrated in Figure 2.2, accumulates incoming signals (currents). This potential "leaks" over time, meaning it gradually decays back to a resting value in the absence of input. When the membrane potential reaches a predefined threshold, the neuron fires a single spike and its potential is reset. The dynamics of the LIF neuron are described by the following differential equation:

$$\tau_m \frac{du}{dt} = -(u - u_{\text{rest}}) + RI(t) \tag{2.1}$$

where u is the membrane potential, u_{rest} is the resting potential, τ_m is the membrane time constant that controls the rate of decay (leak), R is the membrane resistance, and $I(t)$ is the input current. For discrete-time simulations, a common formulation is:

$$v(t) = v(t-1) + \frac{1}{\tau_m}(-v(t-1) + RI(t))\Delta t \tag{2.2}$$

$$s(t) = \begin{cases} 1, & \text{if } v(t) \geq \vartheta \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

If $s(t) = 1$, then $v(t)$ is reset to u_{rest}. Here, $v(t)$ is the membrane potential at timestep t, ϑ is the firing threshold, and $s(t)$ is the output spike.

Current-Based (CUBA) Model: The CUBA model extends the LIF model by incorporating synaptic dynamics, resulting in a second-order system. This allows for a more realistic modeling of how synaptic inputs affect the neuron. The discrete-time dynamics are described by:

$$i(t) = (1 - \alpha_i)i(t-1) + I(t) \tag{2.4}$$

$$v(t) = ((1 - \alpha_v)v(t-1) + i(t) + \text{bias})(1 - s(t-1)) \tag{2.5}$$

$$s(t) = v(t) \geq \vartheta \tag{2.6}$$

where $i(t)$ represents the synaptic current, and $v(t)$ is the membrane potential. The decay factors α_i and α_v control the time constants of the synapse and membrane, respectively. The LIF model can be seen

as a special case of the CUBA model where $\alpha_i = 1$. An example of the dynamics for an input spike train is displayed in Figure 2.3.

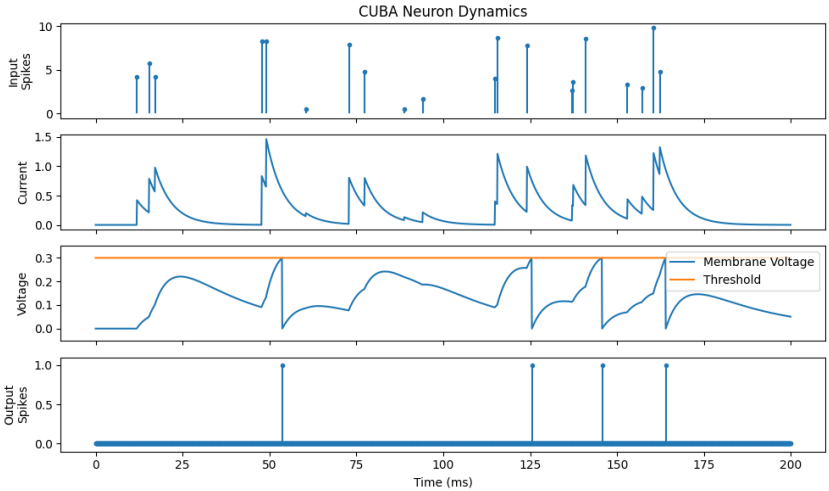


Figure 2.3.: Neuron dynamics of a CUBA neuron when excited with the exemplary spike train as input. Adapted from [Tre25].

Adaptive Leaky Integrate-and-Fire (ALIF) Model: To capture more complex biological phenomena, the ALIF model extends the LIF or CUBA models with mechanisms for spike-frequency adaptation and refractoriness. Adaptation is modeled by a dynamic firing threshold that increases after each spike and then decays back to a baseline value. Refractoriness describes a short period after a spike during which the neuron is less likely to fire again. For a CUBA neuron, the ALIF extension is formulated as:

$$\vartheta(t) = (1 - \alpha_\vartheta)(\vartheta(t-1) - \vartheta_0) + \vartheta_0 + \vartheta_{\text{step}} s(t-1) \tag{2.7}$$

$$r(t) = (1 - \alpha_r) r(t-1) + 2\vartheta(t-1) s(t-1) \tag{2.8}$$

$$s(t) = (v(t) - r(t)) \geq \vartheta(t) \tag{2.9}$$

Here, $\vartheta(t)$ is the adaptive threshold with a baseline ϑ_0, step increase ϑ_{step}, and decay α_ϑ. The refractory term $r(t)$ subtracts from the membrane

potential, with decay α_r. The Dynamics for an example input spike train are illustrated in Figure 2.4.

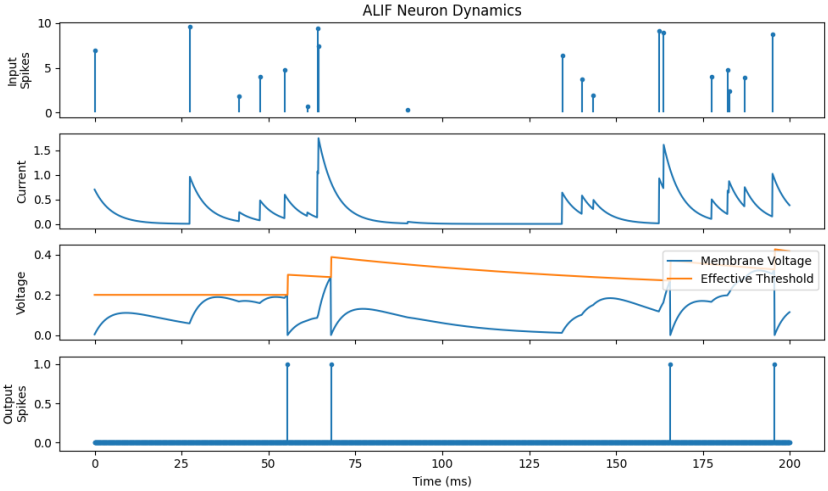


Figure 2.4.: Neuron dynamics of a ALIF neuron when excited with the exemplary spike train as input. Adapted from [Tre25].

Other Models: While the LIF, CUBA, and ALIF models are prevalent in neuromorphic engineering due to their computational efficiency, a variety of other spiking neuron models exist, offering higher levels of biological plausibility. The Hodgkin-Huxley model, for instance, provides a highly detailed description of neuron dynamics by modeling the behavior of ion channels, but its computational complexity makes it impractical for large-scale simulations [34]. The Izhikevich model offers a compromise, using a two-dimensional system of differential equations to reproduce a wide range of firing patterns observed in cortical neurons—such as regular spiking, bursting, and chattering—while being more computationally tractable than the Hodgkin-Huxley model [39].

2.2. Spike Encoding

SNNs process information using discrete, temporally sparse events (spikes), which is fundamentally different from conventional ANNs that operate on continuous-valued data. For maximum efficiency, SNNs should ideally interface directly with event-based sensors that natively produce spike data. However, as such sensors are not yet widely available for all modalities, a crucial step in applying SNNs is the transformation of conventional sensor data into spike trains—a process known as spike encoding. The inverse process, decoding, is also often required to convert the network's output spikes into continuous values for analysis or control of actuators. The choice of an encoding method is a critical design decision, as it determines how information is represented and influences the network's computational efficiency, latency, and performance.

Spike encoding schemes are generally classified into two primary paradigms: rate coding and temporal coding, with further subdivisions as shown in Figure 2.5.

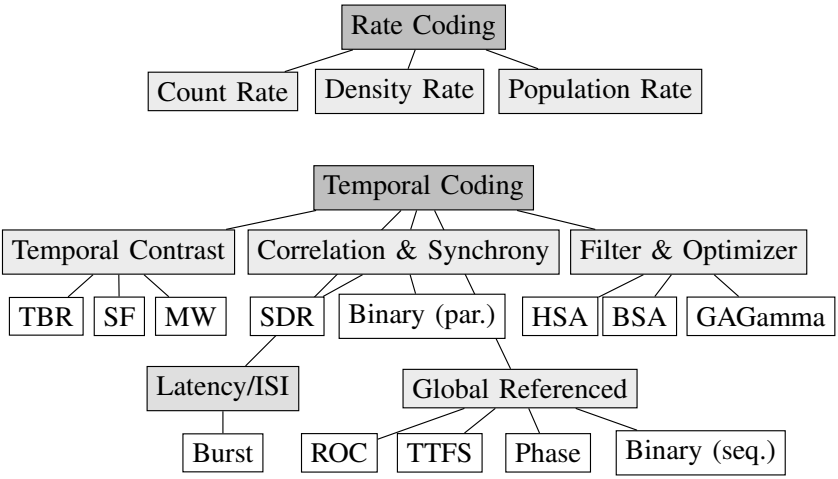


Figure 2.5.: Taxonomy of spike encoding techniques.

Rate Coding

In rate coding, the magnitude of an analog signal is represented by the firing rate of a neuron over a specific time window. A higher input value translates to a higher frequency of spikes. While this method is robust to noise, it can be inefficient, often requiring a larger number of spikes and longer time windows to represent a value with high fidelity, which can increase overall energy consumption.

Temporal Coding

In temporal coding, information is contained within the precise timing of individual spikes. This approach allows for much sparser and more efficient data representations, as a single spike can carry significant information. In general, temporal coding schemes use fewer spikes to encode information than rate-based approaches, which reduces the number of computational steps required and is thus beneficial for energy efficiency. Several distinct algorithms fall under this paradigm, with some of the most common ones described below.

Time-to-First-Spike (TTFS) Coding: is a form of latency coding where the time of the first spike encodes the input value. Typically, a higher input value leads to an earlier spike. This method is very fast as the information is conveyed by a single spike per neuron, allowing for rapid decision-making.

Step-Forward (SF) Encoding: generates spikes whenever the input signal surpasses a dynamically adjusted baseline threshold. When the signal exceeds the threshold above baseline, a positive spike is emitted and the baseline is incremented by the threshold value. Conversely, if the signal drops below the threshold under the baseline, a negative spike is emitted and the baseline is decreased by the threshold value. Finally, if the signal remains within bounds, no spike occurs and the baseline remains constant. This adaptive thresholding mechanism allows the encoder to track signal variations while maintaining sensitivity to changes. A detailed pseudocode implementation is shown in Algorithm A.1 and a visual representation in Figure 2.6.

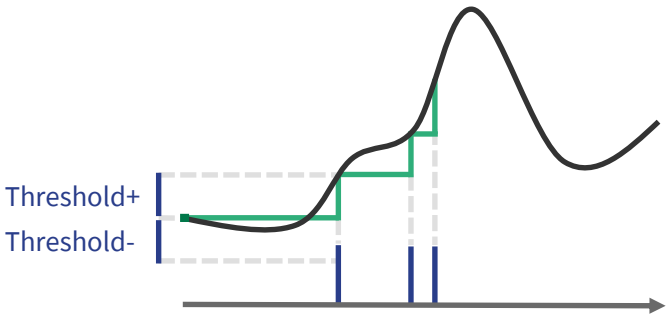


Figure 2.6.: Step-Forward encoding mechanism showing dynamic baseline adaptation and spike generation.

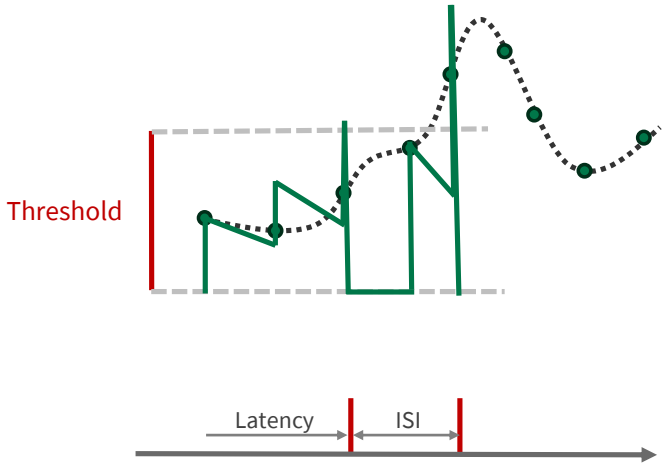


Figure 2.7.: Leaky Integrate-and-Fire encoding showing membrane potential dynamics, integration, and threshold-based spike generation.

LIF Encoding: draws inspiration from biological neuron models. The input signal serves as a current that charges a virtual membrane potential. When this potential exceeds a predefined threshold, a spike is generated and the membrane potential resets. Between spikes, the potential decays exponentially according to a membrane time constant. This approach requires signal normalization to ensure compatibility with fixed threshold and decay parameters, making it particularly suitable for signals that fluctuate around a baseline value. The complete algorithm is presented in Algorithm A.2 and a visual representation in Figure 2.7.

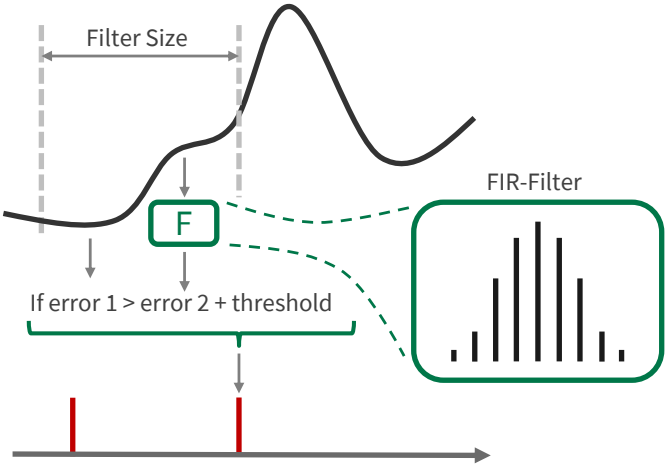


Figure 2.8.: Ben's Spiker Algorithm encoding showing FIR filter operations and spike generation.

Ben's Spiker Algorithm (BSA): employs a more complex approach based on signal reconstruction principles. The algorithm assumes the input signal can be represented through convolution with a Finite Impulse Response (FIR) filter. At each time step, an error term quantifies the difference between the filtered spike train and the original signal. When this error exceeds a threshold, a spike is generated to minimize reconstruction error. This method requires careful parameter tuning but can achieve high reconstruction accuracy for specific signal types. The algorithmic

implementation is detailed in Algorithm A.3 and a visual representation in Figure 2.8.

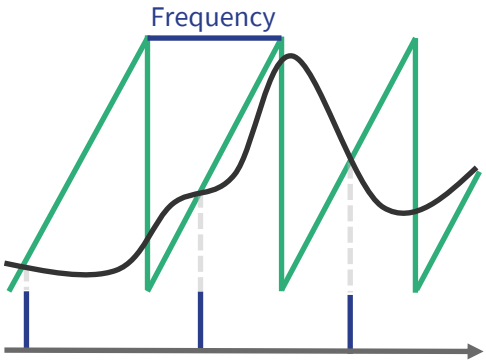


Figure 2.9.: Pulse Width Modulation encoding demonstrating carrier signal comparison and spike generation.

Pulse Width Modulation (PWM) Encoding: compares the input signal against a periodic carrier signal, typically a sawtooth wave. Spikes are generated when the input signal exceeds the carrier signal, with the carrier frequency determining the temporal resolution of the encoding. The method requires signal normalization to ensure proper overlap between input and carrier signals. Algorithm A.4 provides the complete algorithmic description and Figure 2.9 shows a visual representation of the encoding process.

Population Coding: uses a group of neurons to collectively represent a single value, where each neuron is tuned to a specific preferred stimulus value. An input value activates the entire population to varying degrees. The distributed pattern of activity across the population can then be decoded to reconstruct the original value with high precision. This distributed representation also provides robustness against noise or neuron failure. The individual neurons within the population can use either rate or temporal coding schemes.

2.3. Training SNNs

The discrete, non-differentiable nature of the spike generation mechanism in spiking neuron models poses a significant challenge for training SNNs with standard gradient-based optimization methods like backpropagation. The derivative of the spike activation is zero almost everywhere, preventing meaningful gradient flow. To overcome this, several distinct training paradigms have been developed, each with its own trade-offs between biological plausibility, computational efficiency, and performance.

Backpropagation-based Training

The prevailing paradigm for training deep neural networks, Backpropagation (BP), cannot be directly applied to SNNs. The core of the issue lies in the discrete and non-differentiable nature of the spike generation mechanism, as highlighted in Section 2.1. The Heaviside step function used to model spike emission (Equation 2.3) has a derivative that is zero almost everywhere and infinite at the threshold, which prevents the flow of meaningful error gradients necessary for gradient-based optimization. Furthermore, the introduction of a temporal dimension in SNNs means that the error at a given timestep is influenced by past inputs, a dependency that standard BP does not account for.

To address these challenges, training methods for SNNs have been developed that adapt the principles of backpropagation. These methods primarily revolve around two key concepts: surrogate gradients and Backpropagation Through Time (BPTT).

Surrogate Gradients: The problem of the non-differentiable activation function is commonly addressed by using surrogate gradients [68, 114]. This technique involves replacing the derivative of the discontinuous spike activation with a continuous, well-behaved function during the backward pass of training. In the forward pass, the network operates with the standard Heaviside step function, thus preserving the sparse, event-driven computation. During the backward pass, however, the gradient is computed using a smooth approximation, or "surrogate," of the true derivative.

Common choices for surrogate gradient functions include the sigmoid, piecewise linear functions [68], or exponential functions. For instance, a popular formulation for the surrogate gradient $\tilde{\sigma}'(u)$ of the membrane potential u around the firing threshold ϑ is an exponentially decaying function [89]:

$$\tilde{\sigma}'(u) = \alpha e^{-\beta|u-\vartheta|} \tag{2.10}$$

where α and β are hyperparameters that control the shape and scale of the surrogate gradient. This function creates a region around the firing threshold where sub-threshold membrane potentials can still contribute to the gradient, mitigating the "dead neuron" problem where neurons that do not fire cannot learn. While this approximation introduces a bias into the gradient estimation, it has proven effective in enabling end-to-end training of deep SNNs using gradient descent [68, 114].

Backpropagation Through Time (BPTT): To account for the temporal dependencies inherent in SNNs, the training process is often conceptualized as unrolling the network through time, creating a deep computational graph where each time step corresponds to a new layer. This allows for the application of BPTT, an algorithm originally developed for Recurrent Neural Networks (RNNs) [68]. As illustrated in Figure 2.10, the state of a neuron at a given time step t depends on its state at $t-1$. BPTT propagates the error not only backward through the layers of the network (spatially) but also backward in time.

The total gradient for a weight parameter $\mathbf{w}$ is the sum of its gradients computed at each time step over the entire simulation window T:

$$\nabla_{\mathbf{w}}\mathcal{L} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}(t)}{\partial \mathbf{w}} \tag{2.11}$$

where $\mathcal{L}(t)$ is the loss at time t. This process allows the network to learn long-term temporal dependencies between spikes. Algorithms like SLAYER [89] and SuperSpike [114] have successfully combined surrogate gradients with BPTT to train deep SNNs on complex temporal tasks.

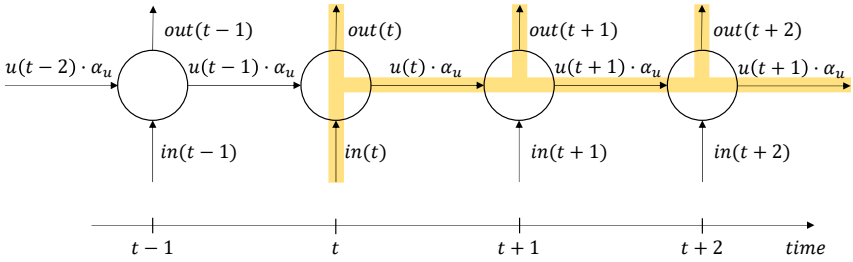


Figure 2.10.: Illustration of Backpropagation Through Time (BPTT) for a single spiking neuron. The temporal dynamics are unrolled, showing how the membrane potential $u(t)$ at each time step is influenced by the potential at the previous time step, modulated by the decay factor α_u. The error gradient is propagated backward through this unrolled graph. Adapted from [Tho24].

EventProp: EventProp is an algorithm that calculates the exact error gradient for continuous-time SNNs without approximations [105]. It uses the adjoint method from optimal control theory to handle the non-differentiability of spike generation. Instead of approximating gradients at each time step, EventProp creates an "adjoint" spiking network to propagate error signals backward in time, computing and propagating errors only at spike events.

EventProp's event-based approach differs from standard BPTT in computational and memory demands. Its backward pass shares the same sparse, event-driven processing as the forward pass, unlike surrogate-gradient-based BPTT, which requires dense matrix-vector products at each time step. Memory usage is determined by the number of spikes, storing only spike times and synaptic currents of firing neurons, rather than the entire state trajectory [105].

Although the method is mathematically exact, it highlights a property of SNNs: the true gradient can become infinite at points where a neuron is close to its firing threshold. Surrogate gradients, in contrast, are an approximation but remain finite. They can be interpreted as a form of implicit regularization that smooths these discontinuities.

Local Plasticity

In contrast to gradient-based methods that require global information to propagate errors, local learning rules adjust synaptic weights based on information available locally at the synapse. These rules are inspired by neurobiological observations of synaptic plasticity, the ability of synapses to strengthen or weaken over time. The most well-known principle is Hebbian learning, often summarized as "neurons that fire together, wire together" [33]. This concept suggests that if a presynaptic neuron repeatedly contributes to the firing of a postsynaptic neuron, the connection between them is strengthened, a phenomenon known as Long-Term Potentiation (LTP). Conversely, connections weaken when neurons fire asynchronously, an effect called Long-Term Depression (LTD) [38].

Spike-Timing-Dependent Plasticity (STDP): A prominent biologically plausible model that formalizes these principles is STDP. STDP is an unsupervised learning rule where the change in synaptic strength is determined by the precise relative timing of pre- and postsynaptic spikes. The magnitude and sign of the weight change, Δw, depend on the time difference $\Delta t = t_{\text{post}} - t_{\text{pre}}$.

As illustrated in Figure 2.11, if a presynaptic spike arrives shortly before a postsynaptic spike ($\Delta t > 0$), causing the presynaptic neuron to contribute to the postsynaptic neuron's firing, the synaptic weight is increased (LTP). If the postsynaptic neuron fires just before the presynaptic neuron ($\Delta t < 0$), it is inferred that the presynaptic spike did not contribute, and the connection is weakened (LTD).

The mathematical formulation for the weight update is given by:

$$\Delta w(\Delta t) = \begin{cases} A_+ e^{-\Delta t/\tau_+}, & \text{if } \Delta t \geq 0 \\ -A_- e^{\Delta t/\tau_-}, & \text{if } \Delta t < 0 \end{cases} \tag{2.12}$$

where A_+ and A_- are scaling factors for potentiation and depression, and τ_+ and τ_- are time constants that define the temporal windows for these changes. The explicit calculation of all spike pairings is computationally expensive. Therefore, implementations often use pre- and postsynaptic trace variables that approximate this function by integrating spike history over time [65].

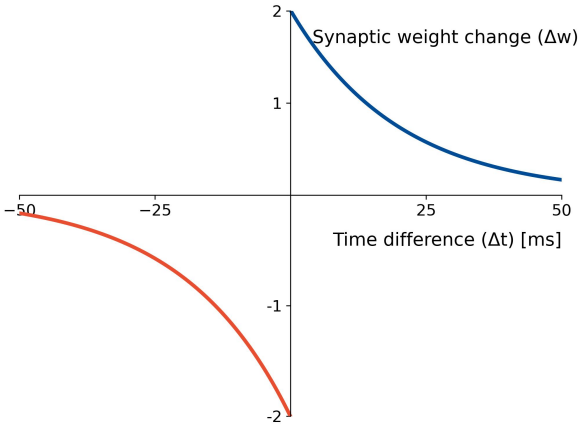


Figure 2.11.: The canonical STDP learning window. The change in synaptic weight (Δw) is a function of the relative timing (Δt) between presynaptic and postsynaptic spikes. Positive Δt leads to LTP, while negative Δt leads to LTD. Adapted from [Sch25].

Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP): While STDP is unsupervised, it can be extended to incorporate reward signals, leading to R-STDP [40]. In this framework, the STDP rule no longer directly modifies the synaptic weight. Instead, it creates a synaptic "eligibility trace", $c(t)$, which marks a synapse as being ready for change. The actual weight update only occurs in the presence of a global neuromodulatory signal, $d(t)$, often analogized to dopamine in the brain, which signals a reward or punishment. The dynamics of the eligibility trace and the neuromodulator are described as:

$$\frac{dc}{dt} = -\frac{c}{\tau_c} + \text{STDP}(\Delta t) \tag{2.13}$$

$$\frac{dd}{dt} = -\frac{d}{\tau_d} + \text{DA}(t) \tag{2.14}$$

where τ_c and τ_d are decay time constants, and $\mathrm{DA}(t)$ represents the externally supplied reward signal. The synaptic weight change is then given by the correlation of the two:

$$\frac{dw}{dt} = c(t)d(t) \tag{2.15}$$

As shown in Figure 2.12, a spike pairing creates an eligibility trace, but the weight remains unchanged until a dopamine signal arrives. This mechanism allows the network to solve the distal reward problem, where the credit for a reward must be assigned to actions that occurred in the past. This algorithm bridges the gap between unsupervised and supervised learning.

E-prop: A more recent development in local learning rules is e-prop [15]. It serves as a more efficient approximation of BPTT. Unlike BPTT, which requires unrolling the network through time and propagating gradients backward, e-prop computes synapse-specific eligibility traces that capture an approximation of the gradient of the loss function. This allows the loss to be propagated only once at the end of a sequence, making the algorithm more computationally efficient and suitable for online, on-chip learning. While the eligibility traces in e-prop are conceptually different from those in R-STDP—approximating a top-down error signal rather than a bottom-up causal link—they similarly enable learning from information that is not strictly local in time.

Evolutionary Algorithms

Inspired by Darwinian principles of natural selection, Evolutionary Algorithms (EAs) offer a gradient-free approach to optimization that is well-suited for the complex and often non-differentiable search spaces associated with SNNs. EAs operate on a population of candidate solutions, iteratively refining them through processes of selection, recombination, and mutation to find optimal network parameters or topologies.

The Evolutionary Optimization for Neuromorphic Systems (EONS) framework employs EAs to automate SNN design and training for neuromorphic hardware [86]. It generalizes evolutionary processes—selection, crossover, and mutation—into a framework applicable to diverse tasks, such as clas-

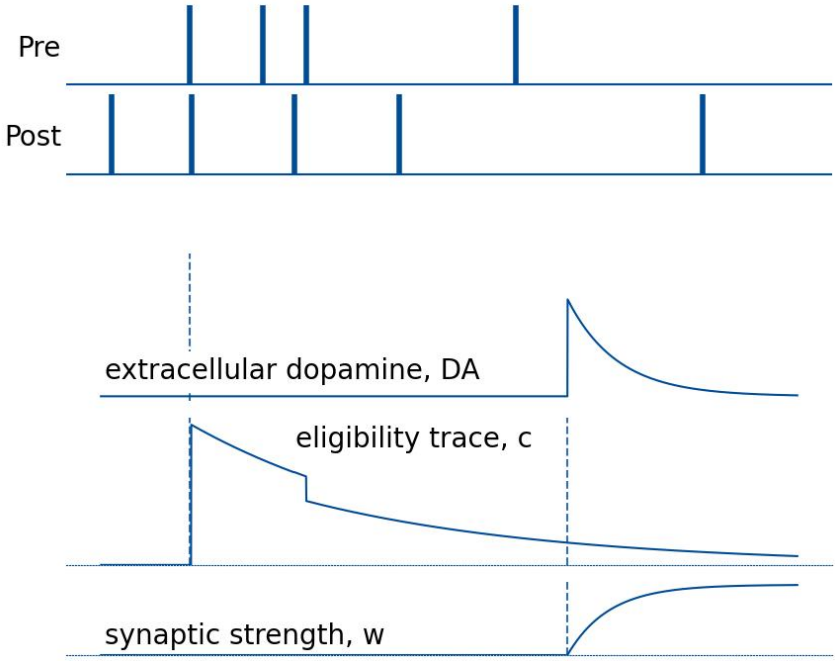


Figure 2.12.: Dynamics of R-STDP at a single synapse. Spike timings (top) generate an eligibility trace (middle). The synaptic strength (bottom) only changes when a global dopamine signal (top) coincides with the eligibility trace. Adapted from [Sch25].

sification and control. Users define a fitness function to evaluate network performance, and EONS optimizes the SNN's parameters and structure to enhance this score. This method allows simultaneous evolution of network topology and parameters, addressing hardware constraints like weight precision and connectivity. Additionally, multi-objective optimization can be achieved by adjusting the fitness function to consider network size or energy consumption, facilitating the development of smaller, efficient networks.

ANN-to-SNN Conversion

Another approach to training SNNs is to first train a conventional ANN and then convert it to a spiking equivalent [82]. This method involves transforming the continuous-valued neurons of the ANN into spiking neurons by mapping activation values to firing rates, often through weight normalization and rescaling. A primary advantage is the ability to leverage mature ANN training frameworks and avoid the complexities of direct spike-based optimization. However, this conversion is not without its challenges. Activation functions that produce both positive and negative outputs, such as the hyperbolic tangent, are difficult to map directly to non-negative firing rates, which can lead to a loss of information and reduced accuracy in the converted SNN.

2.4. Neuromorphic Hardware

To fully leverage the computational advantages of SNNs, specialized hardware is required that departs from the conventional von Neumann architecture. Neuromorphic hardware aims to mimic the brain's structure and function, embracing principles such as massive parallelism, asynchrony, co-located memory and computation, and event-based communication [20]. The primary benefit of this approach is a significant reduction in energy consumption and latency, achieved by eliminating the bottleneck between separate memory and processing units and by computing only when events (spikes) occur [63].

Neuromorphic chips can be broadly categorized into analog, digital, and hybrid designs.

- **Analog systems** implement neuron and synapse dynamics using analog circuits, often operating in the subthreshold regime of transistors where current flow physics closely resembles that of ion channels in biological neurons [61]. This results in high energy efficiency and speed. However, analog circuits are susceptible to noise, temperature variations, and manufacturing inconsistencies, which can affect their reliability.
- **Digital systems** offer robustness, programmability, and benefit from technology scaling. While they consume more power and silicon area per neuron than analog designs, their deterministic behavior makes them easier to work with. Notable large-scale digital platforms include Intel's Loihi [20], IBM's TrueNorth [63], and the SpiNNaker project [37].
- **Hybrid systems** seek to combine the advantages of both paradigms. A prominent example is the BrainScaleS-2 system, which features a custom analog accelerator core for the physical emulation of neuron and synapse dynamics, tightly coupled with a digital processor that handles complex plasticity rules and data processing [75]. This hybrid approach allows for accelerated, energy-efficient emulation of neural dynamics while retaining the flexibility of programmable digital control.

Intel's Loihi 2

As a representative example of a state-of-the-art digital neuromorphic processor, we consider Intel's Loihi 2 chip, a key component of the Kapoho Point research system shown in Figure 2.13. The Loihi 2 architecture is a mesh of 128 asynchronous neuron cores, each with its own local memory, interconnected by a network-on-chip. This design avoids the von Neumann bottleneck and facilitates massively parallel processing. Each core can simulate up to 8,192 programmable neurons, allowing for a wide range of neuron models. The entire chip can thus support up to one million neurons and 120 million synapses. Communication is event-driven, with support for both binary and graded spikes, and the architecture can be scaled

across multiple chips to build larger systems [20]. Such platforms are instrumental for executing SNNs efficiently and are supported by software frameworks like Lava, which facilitate the development and deployment of neuromorphic applications.

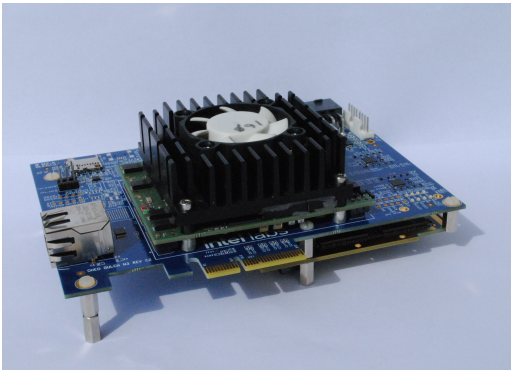


Figure 2.13.: The Kapoho Point system, featuring an Intel Loihi 2 chip on the Mezzanine Card (bottom), which is connected to an Intel Arria 10 FPGA SoM (top) via the Oheo Gulch Baseboard (middle). Figure adapted from [1].

3. A Framework for Energy Minimization in SNNs

3.1. Energy Estimation Paradigm

To estimate the energy usage of neural networks on various platforms, I adopt a methodology inspired by the Nengo framework [14], estimating energy per inference based on synaptic operations and neuron updates:

$$E = E_{\text{synaptic}} + E_{\text{neuron}} \tag{3.1}$$

$$E_{\text{synaptic}} = E_{\text{synop}} \times N_{\text{synops}} \tag{3.2}$$

$$E_{\text{neuron}} = E_{\text{neurop}} \times N_{\text{neuron updates}} \tag{3.3}$$

where E_{synop} is the energy per synaptic operation and E_{neurop} is the energy per neuron update, specific to the target hardware, summarized in Table A.1. The process involves a forward pass of the model, during which, the number of synaptic operations (N_{synops}) is counted. The number of neuron updates ($N_{\text{neuron updates}}$) is the number of neurons N multiplied by the number of timesteps per inference T, which assumes that every neuron is updated at every step. For the synaptic operations, the calculation differs between neuromorphic and traditional hardware:

- Neuromorphic: Because the computation is event-based, the number of synaptic operations is calculated by summing, over all timesteps t, layers l, and neurons n (in layer l) the product of the neuron's spike state $s_{t,l,n}$ (1 if spiking, 0 otherwise) and its number of non-zero

outgoing connections $c_{l,n}$. This assumes the spiking platform can leverage connection sparsity:

$$N_{\text{synops, spiking}} = \sum_{t,l,n} s_{t,l,n} \times c_{l,n} \tag{3.4}$$

where t is timesteps, l layers, and n neurons in layer l.

- Traditional: It is assumed that all synaptic connections are computed every timestep. Total synops are the total number of connections multiplied by timesteps per inference.

Substituting the definition of number of synaptic operations (Equation 3.4) in the synaptic energy definition (Equation 3.2), we get:

$$E_{\text{synaptic}} = E_{\text{synop}} \times \sum_{t,l,n} s_{t,l,n} \times c_{l,n} \tag{3.5}$$

To obtain a simpler objective, we can approximate the synaptic energy, by averaging across the number of neurons in a layer:

$$E_{\text{synaptic}} \approx E_{\text{synop}} \times \sum_{t,l} s_{t,l} \times c_l \tag{3.6}$$

where $s_{t,l}$ is the average spiking probability of a neuron in layer l at timestep t, and c_l is the total number of non-zero outgoing connections in layer l. This can be further approximated, by averaging across layers:

$$E_{\text{synaptic}} \approx E_{\text{synop}} \times \sum_{t} s_t \times N_{\text{connections}} \tag{3.7}$$

where s_t is the average spiking rate of a neuron in the entire network at timestep t, and $N_{\text{connections}}$ is the total number of non-zero connections in the entire network. Finally, this ca be approximated by averaging across the number of timesteps:

$$E_{\text{synaptic}} \approx E_{\text{synop}} \times N_{\text{spikes}} \times N_{\text{connections}} \tag{3.8}$$

where N_{spikes} is the number of times a neuron spikes on average in the entire network during an inference of length T. Substituting this in the total energy equation, we obtain:

$$E \approx E_{\text{synop}} \times N_{\text{spikes}} \times N_{\text{connections}} + E_{\text{neurop}} \times N_{\text{neuron updates}} \tag{3.9}$$

3.2. Energy Minimization

As the approximated synaptic energy depends on the total number of connections ($N_{\text{connections}}$), it scales with the square of the number of neurons $\mathcal{O}(N^2)$, whereas the neuron energy scales linearly with the number of neurons $\mathcal{O}(N)$, since it is the product of total timesteps T and number of neurons N. In practice, therefore, most of the energy consumption is caused by the synaptic operations. If our goal is to minimize energy, we can estimate the total energy by ignoring the second term:

$$E \approx E_{\text{synop}} \times N_{\text{spikes}} \times N_{\text{connections}} \tag{3.10}$$

where E_{synop} is the energy per synaptic operation, N_{spikes} is the number of times a neuron spikes on average in the entire network during an inference of length T, and $N_{\text{connections}}$ is the total number of non-zero connections in the entire network.

By estimating the energy as the product of these three terms, it allows us to conceptually separate the task of minimizing energy consumption into three distinct optimization objectives:

1. E_{synop} - More efficient hardware.
2. N_{spikes} - Sparser neuron activations.
3. $N_{\text{connections}}$ - Sparser network connectivity.

In this framework, a decrease in any of the objectives results in a directly proportional decrease in total energy consumption.

This methodology has several limitations, as data transfer costs to/from the device are not included, and precision (bit-width after quantization) is not considered. Furthermore, we underestimate the energy cost for the

non-neuromorphic devices (x86, ARM, GPU) since we assume that each synop and neuron update is one Multiply-Accumulate Operation (MAC), even though in practice, a neuron update might need more, depending on the implementation. The GPU costs are also estimated for a single sample, averaging across the total batch size, even though in practice, during iteration only single samples are used. Despite this, it provides a comparative estimate of the energy costs across different hardware platforms.

Another limitation arises from the approximation of the number of synaptic operations through averaging of the spiking rate of neurons across the entire network. An example where this could fail is when we reduce the spiking activity of neurons that have no connections. The average spiking rate thus decreases, but since the respective neurons were not connected, the total number of synaptic operations doesn't change, even though the proposed approximation would predict a reduced energy. Nevertheless, the separation of the energy reduction task into three distinct problems allows for clearer objectives and the development of specialized solutions, that can in practice translate to energy improvements.

This work aims for software-level improvements in energy consumption. Since the first term (E_{synop}) is specific to hardware, it will focus on the latter two (Activation Sparsity - N_{spikes}, Connection Sparsity - $N_{\text{connections}}$):

3.2.1. Optimizing Activation Sparsity

Biological neurons are estimated to spike on average with a frequency of 0.16 Hz [52], with maximal firing rates reaching up to 600 Hz [99]. Assuming a maximum biologically possible firing rate of 1000 Hz, we reach an average activation sparsity of 1.60×10^{-4}. This suggests that biological systems have exploited activation sparsity as a way to build very large networks, while still keeping the energy cost low.

SNNs can be employed in a similar fashion to reduce the energy consumption of a neural network. Compared to activation functions in ANNs, SNNs make use of Spiking Neurons (see Section 2.1), which trade of precision of output values (generally binary) for more expressive temporal dynamics (stateful units). This results in inherently sparse activations, thus reducing N_{spikes} and the total energy consumption. The spiking activity can be

reduced further through optimization methods applied during training, which penalize higher spike counts.

Biological systems employ specialized sensors as an interface to the outside world to minimize neuron activity even further. These sensors filter information and encode it in such a way that minimizes the amount of data received by the brain. The vertebrate retina serves as a prime example. It primarily encodes stimuli that change in time over static ones, and detects rapid intensity transients to generate precisely timed spikes [62]. The retina thus encodes only significant temporal differences or events, rather than the absolute, continuous stream of sensory input.

To ensure a similar advantage in artificial systems, input data to the networks should be encoded into events, as a sparse spike train, rather than a continuous stream of numeric values. This can be achieved by use of spike encoding algorithms, optimized for minimal information loss and maximal sparsity (see Section 2.2).

3.2.2. Optimizing Connection Sparsity

The human brain is estimated to contain around 100 billion neurons (8.60×10^{10} [11]) and around 100 trillion synapses (1.00×10^{14} [116]), with around 1000 connections per neuron on average. Considering the total number of possible connections per neuron (around 1.00×10^{11}, in a fully-connected structure), we estimate that the human brain has a connection sparsity of 1.00×10^{-8}. The metabolic cost of a fully-connected network of biological neurons would therefore be as much as 1.00×10^{8} higher than actual values (estimated at 20 W [12]), which would be impossible to achieve with the limited energy budget of biological systems.

Increasing connection sparsity thus becomes a vital component employed by biological systems for decreasing energy consumption. However, it is in part also driven by the resource and spatial constraints in a finite volume of space, allowing only a few number of primarily local synapses, minimizing the cost associated with numerous lengthy connections. This prompted the hypothesis that the spatial limitations of biological systems can be viewed as an optimization framework for sparser network connectivity.

In neural networks, higher connection sparsity can be achieved through pruning. This is typically employed during or after training to eliminate

weights that contribute little to the model's performance. Such methods, however, tend to achieve limited pruning rates, since fully connected architectures start with computing functionality distributed among all connections. As training progresses, this becomes more ingrained, leading to a performance decrease if weights are removed, as small changes in connectivity patterns disturb the precisely optimized balance of each neuron.

By leveraging spatial constraints, sparse connectivity becomes the starting point, leading to more localized computation and allowing for higher pruning rates of further away neurons. This can be achieved by embedding neurons in an euclidean space and making the connection probability or weight dependent on the distance between nodes.

4. Related Work

This chapter reviews the literature relevant to the core research objectives of this dissertation, focusing on software-level strategies for minimizing the energy consumption of SNNs. As established in Chapter 3, the energy footprint of a neural network can be approximated by $E \approx E_{\text{synop}} \times N_{\text{spikes}} \times N_{\text{connections}}$. Accordingly, this review is structured around the two key software-level targets for optimization: activation sparsity (N_{spikes}) and connection sparsity ($N_{\text{connections}}$).

The first two sections survey existing methods for inducing sparsity. Section 4.1 examines techniques for reducing neuron activations in ANNs and SNNs, including a discussion on spike encoding. Section 4.2 reviews approaches for reducing synaptic connections, from traditional pruning methods to more specialized techniques based on spatial embedding.

Section 4.4 outlines the benchmark tasks—predictive maintenance, sleep analysis, neural decoding, and robotic control—used to evaluate the proposed sparsity methods and to validate the research contributions.

Finally, Section 4.3 synthesizes the findings from the literature to identify the specific research gaps that this dissertation aims to address.

4.1. Activation Sparsity

In ANNs, activation sparsity, where a large fraction of neuron activations are zero for a given input, is achieved through several methods. One approach is the use of non-linear activation functions that threshold or clip inputs. The Rectified Linear Unit (ReLU) function, for example, sets all negative inputs to zero, which can lead to a substantial number of zero-valued activations [47, 109]. Variants such as hard-threshold or clipped ReLU extend this by also setting small positive values to zero, further increasing sparsity [47]. Another method involves applying explicit

thresholds during inference to prune low-magnitude activations, as seen in techniques like FA-TReLU, which uses a learned threshold to zero-out activations [47].

Sparsity can also be promoted by modifying the training objective. An ℓ_1 regularization term added to the loss function encourages network weights or activations to become exactly zero [109]. Fine-tuning a pre-trained model with an ℓ_1 penalty on layer outputs has been reported to increase the proportion of zero activations by up to 60% [31]. Other sparsity measures, including Hoyer or ℓ_0 proxy norms, can be used to bias the network toward sparser representations [47]. Similarly, sparse autoencoders employ constraints that limit the average activity in the hidden layer, compelling the network to reconstruct each input using a small subset of active units and thereby learn a sparse basis for the data [71].

Architectural modifications provide a third means of inducing sparsity. Winner-Take-All (WTA) circuits, for instance, use lateral inhibition to ensure that only the neuron with the highest activation (or a top-k subset) fires within a group, forcing all others to zero [106]. Other structural choices, such as bottleneck layers with a reduced number of neurons or the use of sparse convolutional filters, can implicitly restrict the number of neurons that respond to an input.

In contrast to ANNs, SNNs exhibit activation sparsity as an intrinsic property of their computational model. Neurons in an SNN generate discrete, temporally sparse events (spikes) only when their membrane potential exceeds a defined threshold. Consequently, at any point in time, a large proportion of neurons are inactive. This results in activations that are sparse both spatially (across the neuron population) and temporally (across time). Temporal coding schemes can further enhance this effect; for instance, TTFS coding, where each neuron fires at most once per input, produces highly sparse spike patterns [83].

In summary, activation sparsity in ANNs is typically induced via activation functions, regularization penalties, or architectural constraints. For SNNs, however, sparsity is a fundamental characteristic that arises from their event-driven and threshold-based processing.

4.1.1. Spike Encoding

A key factor influencing activation sparsity in SNNs is the method used to convert input data into spike trains, a process known as spike encoding. The choice of an encoding algorithm directly determines the number of input spikes (N_{spikes}), thus affecting not only the network's computational and energy efficiency but also its performance. An effective encoder must balance the trade-off between information preservation and the resulting spike sparsity. This subsection reviews comparative studies of various encoding algorithms and the state of available software frameworks.

Comparative Analyses of Encoding Algorithms Several studies have evaluated spike encoding algorithms across different criteria. Wang et al. [101] implemented SF, BSA, PWM, and Sliding Window (SW) encoding [103] on an Field Programmable Gate Array (FPGA) to assess their suitability for real-time applications. Their findings indicate that while BSA exhibits high power consumption in general, it is efficient for encoding square-wave signals. PWM offers good signal reconstruction with a simple implementation, though its accuracy is dependent on the choice of a curve-fitting algorithm for decoding, which can increase power consumption if non-linear operations are used. SF encoding is also power-efficient and simple to parameterize, but its accuracy can be limited when processing signals with sharp amplitude changes.

Chen et al. [18] also compared SF and PWM, confirming that both can achieve high reconstruction accuracy. They noted that SF becomes more precise with smaller thresholds, at the cost of generating more spikes. Their work highlighted the successful application of these encoders to event-like and Electrocardiogram (ECG) signals, demonstrating their potential for biomedical applications.

Petro et al. [77] proposed a systematic workflow for selecting, optimizing, and validating encoding methods, including SF and BSA. Their results corroborate the versatility and robustness of SF. However, in contrast to Wang et al., they observed high reconstruction errors for BSA on step signals, potentially due to their use of a multiplicative rather than a subtractive threshold.

The applicability of encoders also depends on the downstream task. Yarga et al. [107] evaluated BSA and LIF encoding for a voice-based classification task using a CNN. They found that LIF encoding generally achieved higher classification accuracy with fewer spikes, suggesting its suitability for classification tasks where precise signal reconstruction is not the primary objective. BSA, designed for minimal reconstruction error, was less suited for this task.

Spike Encoding in Software Frameworks The availability of accessible and flexible software tools is crucial for research and development. However, existing open-source spike encoding repositories often cater to specific needs outside of mainstream machine learning workflows. For instance, *SpikeCoding* [25] is designed for real-time robotics control and integration with ROS, while *Spikes* [32] provides basic encoding functionalities but is not integrated with neural network libraries. This landscape reveals a gap for a comprehensive, open-source library that is compatible with widely-used frameworks like PyTorch and supports flexible selection and optimization of encoding algorithms for machine learning research, a gap that this dissertation aims to address.

4.2. Connection Sparsity

Connection sparsity, the reduction of synaptic connections within a neural network, is a critical factor for improving computational efficiency. A primary approach to achieving this is network pruning, where connections are removed from a dense model either during or after training. Magnitude-based pruning, for example, eliminates individual weights with the smallest magnitudes and can remove over 90% of parameters without a significant loss in accuracy [30]. Structured pruning extends this by removing entire components such as neurons or convolutional filters, which simplifies the network architecture and avoids the irregular connectivity patterns that can result from unstructured methods [53]. Pruning is often an iterative process, alternating between connection removal and model fine-tuning to gradually increase sparsity while preserving performance.

An alternative to pruning a dense model is to build sparsity into the network from initialization. One-shot methods such as Single-shot Network Pruning (SNIP) prune a network once before training begins, based on a saliency analysis of connection gradients [51]. Another class of methods grows sparse connectivity during training. For example, Sparse Evolutionary Training (SET) begins with a random graph and dynamically evolves the network's topology by pruning less important connections and adding new ones [64]. Such methods can produce networks with quadratically fewer parameters than their dense counterparts while achieving comparable accuracy. The Lottery Ticket Hypothesis provides further insight into the relationship between initialization and sparsity, suggesting that dense networks contain sparse subnetworks ("winning tickets") that are capable of training to high accuracy from their initial weights alone [30].

Regularization techniques offer another avenue for inducing connection sparsity. By adding penalties to the training objective, these methods encourage weights to approach zero. Variational dropout, for instance, learns a specific dropout rate for each weight, effectively removing many of them by driving their dropout rates to one [35]. Similarly, Bayesian methods that employ sparsity-inducing priors or ℓ_0 regularization also result in a high proportion of near-zero weights.

These methods, extensively reviewed in [35, 51, 53, 64, 110], are broadly applicable to both ANNs and SNNs. A distinct approach, however, draws inspiration from the spatial organization of biological neural systems to impose structural constraints on connectivity.

4.2.1. Spatial Embedding of Neural Networks

In biological systems, the spatial arrangement of neurons is a key determinant of their connection patterns. This principle has inspired the development of spatially embedded neural networks, where connection probabilities and weights are influenced by the distance between neurons in an embedded space, which was explored in recent work. Deb et al. [21] arranged network weights into 2D cortical sheets and applied a topographic loss, achieving higher weight sparsity and parameter efficiency. Elbrecht and Schuman [26] used evolutionary algorithms to optimize connectivity in 2D-embedded SNNs, while Achterberg et al. [8] demonstrated that embed-

ding RNNs in a 3D space with biophysical constraints produced brain-like characteristics, such as modularity. Similarly, Bogdan et al. [16] implemented a 2D SNN on neuromorphic hardware using distance-dependent connection rules, further illustrating the potential of this approach.

4.3. Research Gap

The preceding sections reviewed software-level strategies for inducing activation and connection sparsity. While methods such as pruning and regularization are well-documented, this review identifies several research gaps.

Regarding activation sparsity (N_{spikes}), the use of established ANN architectural components, such as CNNs or Gated Recurrent Units (GRUs), by replacing their conventional activation functions with spiking neurons is an area where the trade-offs between task performance and sparsity benefits require investigation.

Furthermore, to maximize activation sparsity at the network's input, effective spike encoding is essential. While various encoding algorithms are known, a gap exists in software frameworks that allow for the flexible selection and, importantly, the parameter optimization of these algorithms based on user-defined objectives, such as simultaneously achieving high spike sparsity and low information loss, within common machine learning development environments.

Concerning connection sparsity ($N_{connections}$), many existing methods involve pruning connections from an initially dense network. An alternative approach, inspired by biological neural systems where spatial organization influences connectivity [41, 91], is to embed neurons in Euclidean space and make connection properties dependent on distance. Biological systems employ spatial constraints as an optimization framework, achieving remarkable sparsity levels (estimated at 10^{-8} for the human brain - Section 3.2.2) while maintaining computational power.

Several prior studies have explored aspects of spatial embedding in neural networks (Section 4.2.1). However, these approaches have primarily focused on fixed spatial arrangements, predetermined embedding dimensions, or were limited to specific network types. This dissertation explores two

distinct and novel directions for spatial embedding in SNNs that remain underinvestigated:

First, architectures where neuron coordinates in Euclidean space are learnable parameters. Unlike previous approaches that use fixed spatial arrangements, this method allows connection weights to be expressed as a differentiable function of dynamically changing spatial positions, optimizable via gradient-based methods. This enables networks to self-organize their spatial structure based on task requirements, potentially discovering more efficient connectivity patterns than predetermined arrangements.

Second, algorithms for optimizing networks of spatially embedded spiking neurons positioned within fixed spatial grids, with a focus on comparing the impact of different embedding dimensionalities on network performance and energy efficiency. Previous works have typically employed a single embedding approach without comparative analysis. By generating connections based on distance-guided probability and strength, and co-optimizing connectivity with spiking neuron parameters, this approach creates inherently sparse yet powerful networks specifically tailored for SNN computation.

These research directions aim to leverage spatial constraints as a foundational design principle rather than an afterthought, potentially achieving connection sparsity levels closer to biological systems while maintaining or improving computational capabilities for neuromorphic applications.

4.4. Application Tasks

To ground the investigation of sparsity-inducing techniques in practical applications, this dissertation defines and addresses a set of diverse benchmark tasks. These tasks, spanning industrial monitoring, biomedical signal processing, and robotics, serve as testbeds for evaluating the effectiveness of the proposed methods in realistic scenarios. The following subsections provide a brief overview of the relevant literature for each task and explain how each task is defined.

4.4.1. Vibration-Based Predictive Maintenance

Predictive Maintenance (PM) is a key strategy in Industry 4.0 for reducing operational costs and preventing unexpected downtimes by monitoring the health of machinery. Analyzing vibration signals from rotating machinery, such as pumps and motors, is a particularly effective method for detecting signs of wear, misalignment, or other faults [98]. However, traditional PM approaches that rely on transmitting high-resolution sensor data to the cloud for analysis incur substantial energy and storage costs, making them less suitable for battery-powered edge devices [69]. Shifting data processing to the sensor edge offers a path toward more energy-efficient solutions.

The survey performed on Neural Network (NN)-based approaches for vibration-based PM reveals several common trends and challenges [VNF+24]. First, there is a lack of a standardized benchmark dataset, leading many studies to rely on custom-generated data or, most commonly, the Case Western Reserve University (CWRU) bearing dataset [57]. Second, data preprocessing typically involves a transformation from the time domain to the frequency domain, using methods like the Fast Fourier Transform (FFT), Short-Time Fourier transform (STFT), or Gammatone Filterbanks (GFBs) [74]. Third, while ANN research has explored a variety of architectures, including CNNs and Long Short-Term Memory (LSTM) Networks, SNN applications have predominantly used shallow, feed-forward networks, with some exceptions using Long Short-Term Spiking Neural Networks (LSNNs) or reservoir computing [112, 24, 45]. The training is mostly supervised, often using surrogate gradient methods or ANN-to-SNN

conversion techniques. A significant gap in the literature is the limited focus on hardware implementation and concrete energy estimations for edge devices [VNF+24, 23, 13].

This dissertation addresses this gap by defining a complex, multi-task PM problem for an industrial Progressing Cavity Pump (PCP). As detailed in Section 5.1.4, the task involves using 3-axis vibration data for simultaneous regression of operational parameters (flow, pressure, pump speed) and multi-label classification of critical machine states (normal, overpressure, cavitation). This use case serves as a testbed for evaluating the energy efficiency of a recurrent SNN architecture designed for on-device deployment, linking the concepts of activation and connection sparsity to practical industrial application.

4.4.2. Sleep Stage Classification from ECG Signals

The accurate classification of sleep stages and the detection of respiratory events are important for diagnosing sleep disorders such as Obstructive Sleep Apnea (OSA). It is estimated that 936 million adults globally have mild to severe OSA, a condition which can lead to severe health complications if untreated [55]. The clinical gold standard for diagnosis, Polysomnography (PSG), is resource-intensive, requiring complex equipment and expert manual scoring. Methods based on single-lead ECG signals have been investigated as an alternative, as ECG data contains information about respiratory and cardiac patterns correlated with sleep stages and apnea events [92].

The literature shows a focus on using machine learning for automated ECG-based sleep analysis. Deep learning models, such as Deep Convolutional Recurrent (DCR) networks and LSTMs, have been applied to multi-class sleep staging tasks [96, 104]. While much of the research has concentrated on ANNs, the use of SNNs is a more recent field of study. Recent work has explored their application for both sleep stage classification and apnea detection, given their suitability for time-series data [42, 95].

The task of sleep stage classification from ECG signals is typically defined as a multi-class classification problem. Models are trained on labeled time-series data to predict sleep stages, which commonly include Wake, N1, N2, N3, and REM sleep. The task can be extended to a multi-label problem

by also requiring the detection of respiratory events such as Obstructive Apnea (OA) and Obstructive Hypopnea (OH). This problem is often benchmarked using public datasets like the ISRUC-Sleep dataset, which contains single-lead ECG recordings with expert annotations for both sleep stages and respiratory events [44].

This dissertation contributes to this task by developing a hybrid CNN-SNN-RNN architecture, detailed in Section 5.1.2, where LIF neurons replace conventional activation functions in a CNN. The objective is to investigate how this architectural modification impacts activation sparsity (N_{spikes}) and overall computational load while aiming to maintain or exceed the classification performance of an equivalent ANN model.

4.4.3. Neural Decoding for Brain-Machine Interfaces

Brain-Machine Interfaces (BMIs) are a promising technology for restoring motor function in individuals with paralysis by decoding neural activity to control prosthetic devices [49]. The development of wireless, fully implantable intra-cortical Brain-Machine Interfaces (iBMIs) is a key area of research aimed at improving patient mobility and reducing infection risk [54, 90]. However, these devices operate under strict power and thermal constraints, which limits their wireless data bandwidth and necessitates energy-efficient, on-device signal processing [80]. SNNs are well-suited for this role due to their inherent event-driven and sparse computational nature, making them promising candidates for low-power neural decoders [117, 81].

The task of neural decoding for motor control involves translating recorded neural signals—typically spike trains from the motor cortex—into continuous movement commands, such as hand or cursor velocity. This problem is often benchmarked using publicly available datasets like the "Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology" dataset [70], which contains spike recordings from primates performing reaching tasks. Frameworks like NeuroBench provide standardized metrics for evaluating models on both decoding accuracy (e.g., R^2 score) and computational resource usage (e.g., memory footprint, synaptic operations) [108].

This dissertation addresses this challenge by focusing on the trade-offs between accuracy and energy efficiency in hybrid SNN-RNN architectures [VKKB24], detailed in Section 5.1.3. A central aspect of this work is the development and analysis of a Spiking Gated Recurrent Unit (sGRU) and its comparison with conventional GRU and LIF recurrent units to investigate the impact on activation sparsity (N_{spikes}) and computational load.

Furthermore, this work is extended to include real-time constraints critical for practical BMI applications, such as execution latency below 100 ms and an update rate of at least 10 Hz [KVKB25]. This involves developing real-time capable architectures and applying model compression techniques to meet the demands of resource-constrained embedded systems, thereby bridging the gap between theoretical model performance and practical, hardware-efficient deployment.

4.4.4. Control Tasks

Continuous control tasks, particularly those in robotics and simulated physical environments, serve as important benchmarks for evaluating the capabilities of NNs. These tasks, often formalized as Reinforcement Learning (RL) problems, require an agent to learn a policy that maps environmental observations to actions to maximize a cumulative reward signal. While deep RL has achieved significant success using ANNs [85], the training of SNNs for control remains an active area of research. Gradient-based methods can be challenging to apply due to the non-differentiable nature of spiking neurons, and often only optimize synaptic weights while leaving other neuron parameters fixed [113].

Gradient-free optimization methods, such as EAs, offer an alternative by directly evolving all aspects of an SNN, including network topology and individual neuron parameters [79]. Another line of research explores the use of local, neuro-inspired learning rules such as R-STDP, which adjust synaptic weights based on local activity and a global reward signal [40]. These approaches are particularly relevant for developing energy-efficient controllers for edge devices, where on-chip learning and adaptation are desirable.

This dissertation investigates the use of EAs to optimize spatially embedded SNNs for continuous control tasks from the Gymnasium [94] MuJoCo suite [93], as detailed in [VSSB25] and Section 5.2.2. The study examines how spatial constraints, implemented by arranging neurons in grids of varying dimensionality (0D, 1D, 2D, 3D), affect performance, energy efficiency, connection sparsity ($N_{\text{connections}}$), and activation sparsity (N_{spikes}). This approach provides a framework for designing sparse and computationally efficient SNNs for control applications.

5. Methods for Inducing Sparsity in SNNs

This chapter presents the proposed methods for optimizing energy consumption in SNNs through software-level improvements targeting both activation sparsity and connection sparsity. The methods are organized according to the energy estimation framework (Equation 3.10) introduced in Chapter 3 ($E \approx E_{\text{synop}} \times N_{\text{spikes}} \times N_{\text{connections}}$), focusing on reducing N_{spikes} and $N_{\text{connections}}$.

5.1. Optimizing Activation Sparsity

Activation sparsity reduction targets the N_{spikes} term in our energy model by minimizing the average firing rate of neurons throughout the network. This is achieved through two complementary approaches: optimized spike encoding at the input layer and integration of spiking neurons within established ANN architectures.

5.1.1. Spike Encoding for Sparse Input

A central advantage in deploying SNNs for machine learning is the reduced energy consumption, which, among others, is also linked to the sparsity of input spike trains. Traditional datasets consist of real-valued input vectors, not spike trains, necessitating efficient encoding methods that maximize spike sparsity while preserving information. This subsection presents the development and evaluation framework for four temporal encoding algorithms that transform continuous signals into sparse spike representations suitable for neuromorphic processing, as presented in [VSS+25]. The overarching goal of these methods is to enable energy-efficient SNN operation by minimizing the information loss during signal encoding, while also introducing sparsity.

Encoding Algorithm Implementation Four distinct temporal encoding methods were implemented and evaluated: LIF, SF [43], PWM [10], and BSA [84], with their pseudocode given in the Appendix (Algorithm A.1, A.2, A.3, A.4). The fundamental principles of these algorithms have been detailed in Section 2.2. This section focuses on the experimental framework used for their evaluation and the software architecture developed for their practical application.

Experimental Evaluation Framework To assess the performance characteristics of each encoding method, an evaluation framework was developed that encompasses reconstruction accuracy, computation time, energy consumption, and spike sparsity metrics.

Evaluations were conducted on an MCX-N947-EVK development board from NXP Semiconductors, featuring dual Arm Cortex-M33 cores operating at 12 MHz with DCDC core voltage of 1.2V. It thus provides realistic constraints for low-power embedded applications.

Five primary evaluation criteria were established:

- **Reconstruction Error**: Mean Squared Error (MSE) between original and decoded signals
- **Spike Sparsity**: Percentage ratio of generated spikes to total signal length (16384 time steps)

- **Encoding Time**: Computational time for spike generation on the embedded device
- **Power Consumption**: Absolute power consumption and dynamic power consumption (with baseline subtracted)
- **Energy Consumption**: Dynamic energy consumption

The evaluation methodology involved measuring absolute power consumption over periods of ten hours and subtracting a baseline no-operation (NOP) power measurement to isolate the dynamic power consumed specifically by each encoding method. Energy consumption was calculated as the product of dynamic power and encoding time.

Four distinct signal types were used to evaluate algorithm performance:

- **Vibration Signals**: High-frequency oscillatory data
- **Trended Signals**: Gradually changing values with underlying directional movement
- **Rectangular Signals**: Step functions with abrupt transitions between discrete levels
- **Sinusoidal Signals**: Smooth periodic waveforms

Figure 5.1 illustrates representative examples of each signal type used in the evaluation, demonstrating the characteristics that the encoding algorithms must handle.

Parameter Optimization Strategy Each encoding method was optimized over 500 trials to minimize reconstruction error (MSE between original and reconstructed signals). Parameter selection, encoding, decoding, and error computation were performed iteratively. Figure 5.2 illustrates this encoding-decoding-optimization process. Optuna [9] was used for optimization: random sampling for most methods, and TPESampler [72] for BSA due to its three interdependent parameters.

Framework Architecture and Implementation To enable practical application of spike encoding methods for minimizing N_{spikes} in the energy estimation framework, we developed an open-source PyTorch-compatible

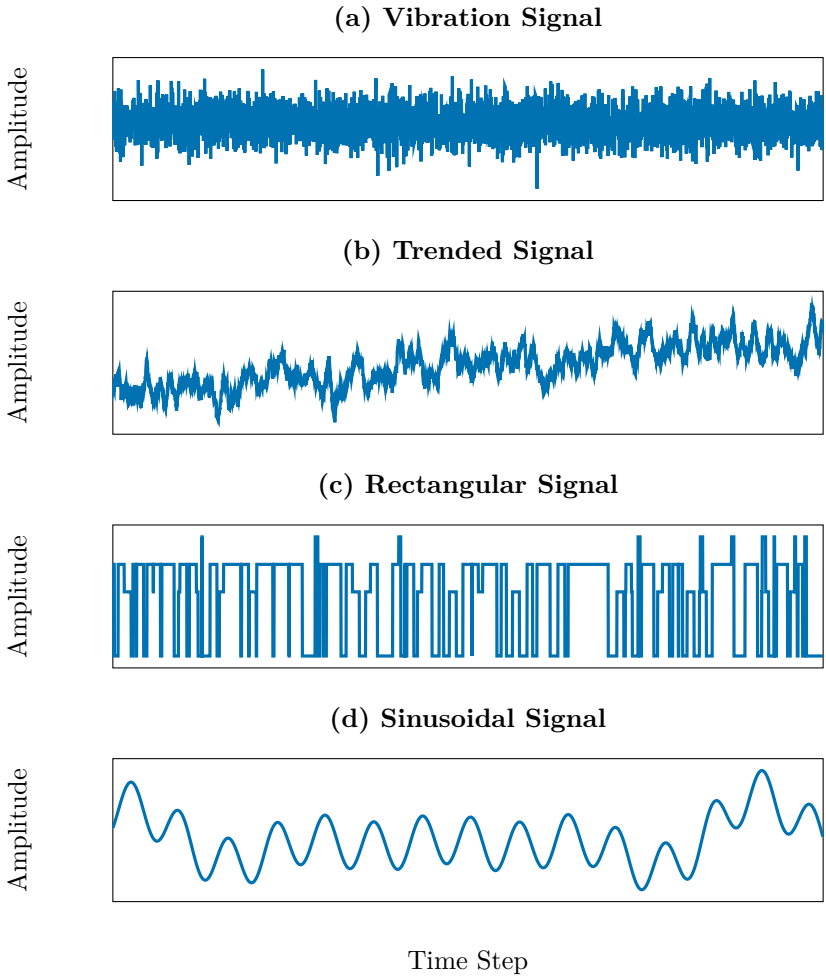


Figure 5.1.: Four different signal types used for encoding algorithm evaluation: (a) Vibration signal showing high-frequency oscillations, (b) Trended signal with gradual directional changes, (c) Rectangular signal with abrupt step transitions, and (d) Sinusoidal signal with smooth periodic variations.

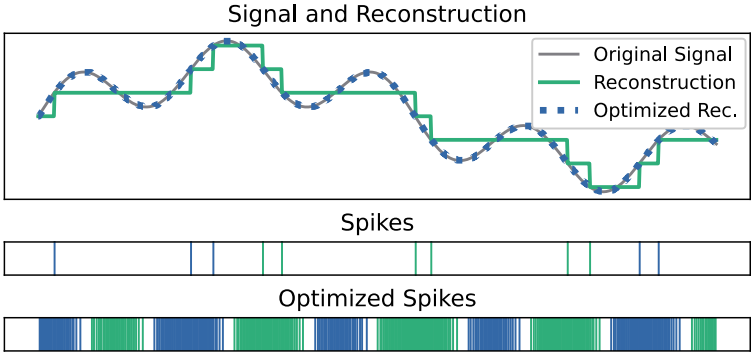


Figure 5.2.: Encoding-decoding-optimization concept showing parameter tuning effect for achieving optimal reconstruction.

framework [97] designed for seamless integration with machine learning workflows. This framework incorporates several key features. It includes built-in parameter optimization to ensure minimal information loss during conversion. The framework is compatible with PyTorch, utilizing native tensor operations for integration with PyTorch datasets. Its architecture is extensible, allowing for the addition of new encoding methods.

The framework is based on the Converter abstraction, where each Converter object implements encoding, decoding, and optimization methods. This design enables each encoding method to achieve optimal reconstruction accuracy through automated hyperparameter tuning, eliminating manual parameter selection while allowing for arbitrary optimization criteria, including co-optimization of sparsity.

For example, the SF usage shown in Figure 5.3 demonstrates the straightforward application. LIF, PWM, and BSA encoding follow analogous interfaces, ensuring consistency across different encoding methods.

The optimization capability, illustrated in Figure 5.4, enables automatic parameter tuning through the optimize() method. This feature is essential for energy-efficient SNN applications, ensuring that encoding parameters are optimized for the specific characteristics of input signals.

```
import torch
from encoding.step_forward_converter import StepForwardConverter

signal = torch.tensor([0.1, 0.3, 0.2, 0.4, 0.8])

converter = StepForwardConverter()
spikes = converter.encode(signal)
```

Figure 5.3.: Using the StepForwardConverter to encode a signal.

```
# ... same as above ...
optimal_threshold = converter.optimize(signal)
optimized_converter = StepForwardConverter(optimal_threshold)

spikes = optimized_converter.encode(signal)
reconstructed_signal = optimized_converter.decode(spikes)
```

Figure 5.4.: Optimization and decoding for signal reconstruction.

The framework also includes specialized components for diverse application scenarios.

The GymnasiumEncoder extends rate encoding methods with utility functions tailored for reinforcement learning environments, enabling direct integration with Gymnasium [94] workflows as shown in Figure 5.5.

The BinEncoder implements Gaussian Receptive Field (GRF)-based population coding to transform single values into multi-dimensional spike representations, supporting applications requiring more biologically plausible encoding methods, as illustrated in Figure 5.6.

This framework addresses a gap in the SNN ecosystem by providing an interface for converting numeric datasets into sparse spike trains with minimal information loss, supporting the goal of minimizing neuron activity (N_{spikes}) in energy-efficient neuromorphic applications.

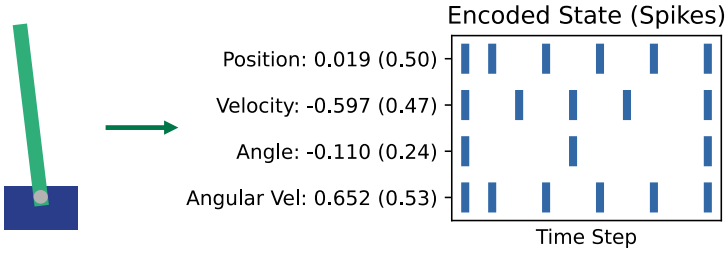


Figure 5.5.: Gymnasium environment integration showing observation encoding for reinforcement learning applications.

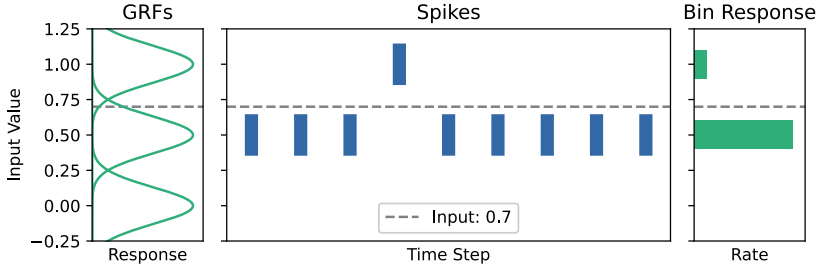


Figure 5.6.: Gaussian Receptive Field population coding showing distributed spike representation generation.

5.1.2. Spiking Neurons in CNNs: Sleep Analysis

To leverage the inherent activation sparsity of spiking neurons within established convolutional architectures, we developed a hybrid approach that integrates LIF neurons into traditional CNN frameworks [BVH+24], applied to a 1D convolutional layer. This method targets the N_{spikes} term in our energy estimation framework by replacing dense activation functions with temporally sparse spiking mechanisms while preserving the computational advantages of convolutional operations.

Dataset and Preprocessing This methodology was applied to the task of sleep stage classification using the ISRUC-Sleep dataset [44]. This dataset provides single-channel ECG signals recorded at 200 Hz, along with expert annotations for sleep stages (Wake, N1, N2, N3, REM) and respiratory events (Obstructive Apnea (OA), Obstructive Hypopnea (OH)). For the network implementation, input ECG segments of 6000 time steps (30 seconds) were used. Preprocessing of the ECG data was consistent for both SNN and ANN models to ensure comparability. It involved z-score normalization, where data points with a z-score exceeding 5 were considered outliers and replaced using linear interpolation. Subsequently, the `ecg_clean` function from the NeuroKit2 library [59] was applied to further enhance signal quality, followed by standardization to zero mean and unit variance.

Architectural Design and Implementation The hybrid architecture integrates LIF neurons into a Deep Convolutional Recurrent (DCR) design, maintaining the core CNN structure—including convolutional layers, batch normalization, and pooling operations—while replacing traditional activation functions like ReLU with LIF neurons. Figure 5.7 illustrates this integration within the convolutional layers.

The convolutional backbone consists of three blocks:

- The first block features a 1D convolutional layer with 60 filters (kernel size 50).
- The second block employs a 1D convolutional layer with 30 filters (kernel size 30).

- The third block uses a 1D convolutional layer with 10 filters (kernel size 20).

Each convolutional layer is succeeded by batch normalization, a LIF activation layer, and a max-pooling layer (pool size 2). The output from this backbone, after undergoing the temporal enhancement strategy described below, is summed and then processed by a recurrent section. This section comprises two GRU layers with layer normalization, designed to capture temporal dependencies between consecutive segments. A final fully connected layer with a softmax activation function performs the classification.

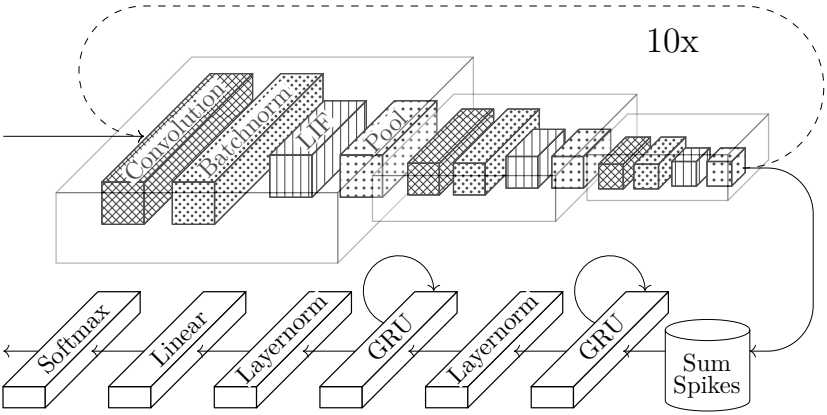


Figure 5.7.: Spiking neural network architecture showing the integration of LIF neurons in convolutional layers. Different textures represent convolution, batch normalization, LIF activation, and max-pooling operations. The dotted line indicates the temporal enhancement strategy where data passes through the backbone multiple times. Figure adapted from [BVH+24].

Temporal Enhancement Strategy A challenge in applying spiking neurons to non-temporal dimensions of convolutional layers is generating sufficient temporal dynamics for the spiking mechanisms. To address this, the convolutional backbone processes the same input multiple times within a single forward pass. Each iteration's output is stored, and after a

set number of passes, these outputs are summed before being fed to the subsequent network stage. This strategy allows LIF neurons to integrate information over several iterations, increasing the likelihood of spike events from subthreshold signals, while preserving overall activation sparsity.

LIF Neuron Implementation and Training The LIF neurons were implemented using the snnTorch library [28], which facilitates the integration of spiking neuron models within PyTorch [73]. The membrane potential decay rate was a key hyperparameter, optimized using Optuna [9], resulting in a value of 0.5261. The number of repetitions for the temporal enhancement strategy was also optimized, with 10 repetitions found to be optimal for maximizing performance. Training utilized the Adam optimizer [46] with a learning rate of 1.4857e-05 and surrogate gradients to handle the non-differentiable nature of the spiking mechanism. PyTorch Lightning [29] was used to structure the training process, incorporating early stopping.

The replacement of ReLU activations with LIF neurons directly contributes to reducing N_{spikes}, as LIF neurons only activate (spike) when their membrane potential reaches a threshold, leading to sparser activations compared to ANNs where activations are typically dense. However, introducing the temporal enhancement strategy—where the convolutional backbone is repeated multiple times per input—creates a trade-off: while this repetition enables sufficient temporal integration for learning and can improve classification accuracy, it also increases the total computational workload proportionally. Thus, the number of backbone repetitions is a critical hyperparameter for balancing energy efficiency and model performance. Selecting this parameter carefully ensures that the advantages of sparse spiking activations are not offset by computational overhead, allowing for application-specific tuning.

5.1.3. Spiking Neurons in RNNs: Neural Decoding

While spiking neurons in convolutional layers enable activation sparsity, incorporating them into recurrent layers further reduces N_{spikes} through event-driven, sparse computation. The challenge lies in preserving the sophisticated gating mechanisms that make recurrent units effective for sequence modeling while introducing the sparse, event-driven dynamics characteristic of spiking neurons.

Spiking Gated Recurrent Unit (sGRU) To capture complex temporal dependencies while preserving spiking dynamics, in [VKKB24] we introduce the sGRU, which fuses the gating mechanisms of traditional GRUs with the event-driven nature of LIF neurons. The sGRU maintains the selective memory capabilities of traditional GRUs while introducing sparsity through spiking mechanisms by replacing the gate computation with LIF neurons:

$$\mathbf{r}_t = \text{LIF}(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1}) \tag{5.1}$$

$$\mathbf{z}_t = \text{LIF}(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1}) \tag{5.2}$$

$$\tilde{\mathbf{h}}_t = \text{LIF}(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h((1 - \mathbf{r}_t) \odot \mathbf{h}_{t-1})) \tag{5.3}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{5.4}$$

where $\mathbf{r}_t$, $\mathbf{z}_t$, $\tilde{\mathbf{h}}_t$, and $\mathbf{h}_t$ denote reset gate, update gate, candidate hidden state, and hidden state at time t, respectively. $\mathbf{W}_r$, $\mathbf{W}_z$, $\mathbf{W}_h$, $\mathbf{U}_r$, $\mathbf{U}_z$, and $\mathbf{U}_h$ are learnable parameters. $\mathbf{x}_t$ denotes the input. By applying spiking dynamics to the gates, the sGRU maintains the memory mechanisms of GRUs while introducing sparsity in computations and the memory state.

Neural Decoding Task and Dataset The neural decoding approaches were validated using the "Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology" dataset [70], which contains spike recordings from two Macaque monkeys (Indy and Loco) while reaching for targets on an 8x8 grid. For Indy, recordings were made from the left hemisphere while reaching with the right arm, while for Loco, the right hemisphere was recorded during left arm movements. Most sessions

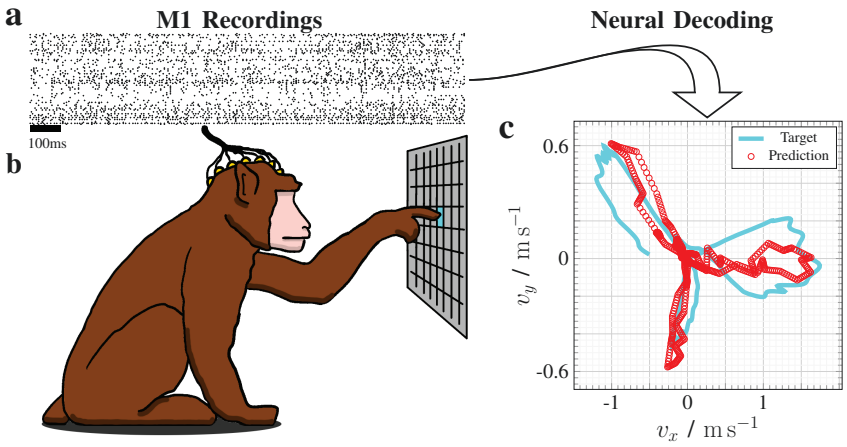


Figure 5.8.: Visualization of the Primate Reaching dataset and the related neural decoding task. **b** illustrates the primates reaching for a target while their brain activity is recorded. **a** shows some resulting spike trains from the motor cortex recordings. **c** presents the prediction of our SNNs for one of the sequences within the recordings and the respective target. Figure adapted from [KVKB25].

included 96 input channels from the primary motor cortex (M1), with some sessions also including 96 additional channels from the somatosensory cortex (S1). The task involved decoding the spike recordings into fingertip velocities projected on a two-dimensional plane, as illustrated in Figure 5.8. Six specific recordings were used for evaluation: three from Indy (with M1 data) and three from Loco (with both M1 and S1 data).

Neural Decoding Architecture The neural decoding application uses a hybrid architecture designed for predicting cursor movement velocities from multichannel neural recordings, as shown in Figure 5.9. This architecture is motivated by the observation that primate reaching movements can be approximated by target-directed ballistic movements with fewer keypoints rather than requiring fine-grained continuous adjustments. This insight enables an efficient architecture that reduces computational requirements while maintaining high decoding accuracy. The architecture consists of:

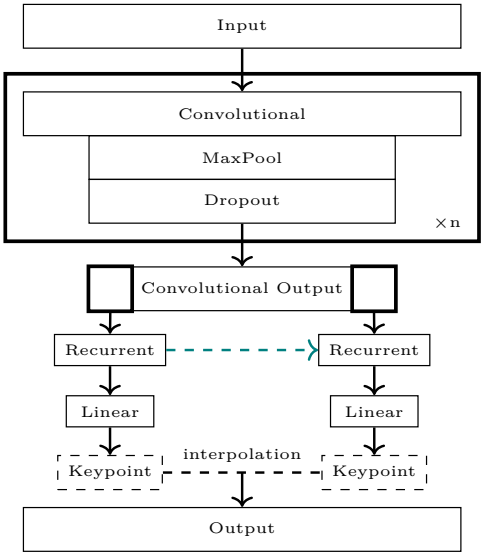


Figure 5.9.: Hybrid network architecture for neural decoding, combining temporal convolutions for dimensionality reduction, recurrent processing for temporal feature integration, and interpolation for output sequence reconstruction. Figure adapted from [VKKB24].

1. **Temporal Convolution**: The input sequence (length 1024, representing 4.096 seconds of neural recordings sampled at 4 ms intervals) is processed through multiple convolutional blocks to reduce the sequence length to a desired number of keypoints and extract relevant temporal features.
2. **Recurrent Processing**: The compressed sequence is processed by one of three types of recurrent units—GRU, sGRU, or LIF—to integrate temporal information. For LIF networks, recurrent weights are employed to maintain temporal context.
3. **Output Layer**: A fully connected layer maps the recurrent unit outputs to predict velocity keypoints.
4. **Interpolation**: Linear interpolation is applied between the predicted keypoints to reconstruct the full-length output sequence (1024 steps).

Two model size variants were implemented in [VKKB24] for different optimization objectives:

- **Track 1 Models**: Optimize for maximum decoding accuracy with three convolutional blocks (32 channels, kernel sizes 3, 6, 12), 8-step interpolation with 127 keypoints, and recurrent blocks of size 64.
- **Track 2 Models**: Co-optimize for accuracy and efficiency with two convolutional blocks (10 channels, kernel size 3), 4-step interpolation with 257 keypoints, and recurrent blocks of size 20.

Max pooling layers use a kernel size and stride of 2 in both model variants.

Real-Time Implementation In [KVKB25], we build on the neural decoding architecture presented in [VKKB24] to enable real-time deployment. For effective neural prosthetic control, the system must maintain a maximum latency of approximately 100ms—the delay between brain impulse and voluntary muscle contraction [48]—and support execution rates of at least 10 Hz [87].

To meet these requirements, we implemented a buffering approach for convolutional layers that computes only the necessary data for each new prediction without redundant calculations, formalized in Algorithms A.5 A.6, A.7, and A.8. The implementation determines three critical buffer sizes at each convolutional layer (illustrated in Figure 5.10):

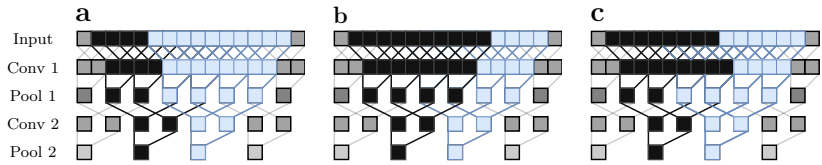


Figure 5.10.: Buffer size calculations for real-time implementation: **a** buffer size per keypoint, **b** buffer size for new data updates, **c** buffers required to process incoming data. Figure adapted from [KVKB25].

a) **Keypoint Buffer**: The effective receptive field required in each layer to compute one output keypoint

b) **Update Buffer**: The minimum new data needed to compute the next keypoint

c) **Processing Buffer**: The size needed to compute the new data in subsequent layers

Real-Time Hyperparameter Optimization The real-time implementation required careful optimization of hyperparameters to balance temporal constraints with decoding accuracy:

- **Execution Rate**: Controlled by the interpolation size and convolutional layers. An interpolation size of 4 with two convolutional layers achieves 62.5 Hz, while 8 steps with three layers provides 31.25 Hz.
- **Latency Control**: Primarily affected by kernel size configuration. Using two convolutional layers with kernel sizes 9 and 18 achieves 96ms latency, meeting the sub-100ms requirement.
- **Architecture Parameters**: Key parameters include the number of convolutional channels (N_C: 10-40), kernel sizes (k: selected for latency requirements), number of LIF layers (N_L: 1-3), and LIF units per layer (N_{LIF}: 20-64).

Real-Time Model Variants Three model variants were developed in [KVKB25] for different deployment scenarios:

- **BMnet (Benchmark Network)**: Optimizes solely for accuracy using three convolutional layers with kernel sizes 31, 62, and 124, achieving a receptive field of 652 points but with 1308ms latency.
- **RTnet (Real-Time Network)**: Balances accuracy with real-time constraints using two convolutional layers with kernel sizes 9 and 18, achieving 96ms latency with a 46-point receptive field.
- **sRTnet (small Real-Time Network)**: Co-optimizes accuracy, memory footprint, and computational demand through fixed-point quantization (1 integer bit, 7 fractional bits for weights; 1 integer bit, 4 fractional bits for activations, plus sign bit). Also uses two convolutional layers with kernel sizes 9 and 18, achieving 96ms latency with a 46-point receptive field.

All models use 40 convolutional channels and 2 LIF recurrent layers with 56 units each.

Network Compression for Resource-Constrained Devices To address the limited computational resources of iBMI devices, we applied compression techniques during training of all models to reduce footprint and computational demand:

- **Spike Regularization**: Adds a penalty term ($\lambda_S = 2.87 \times 10^{-3}$) to the loss function based on total spike count, encouraging sparser activations.
- **Weight Regularization**: Applies L1 regularization ($\lambda_W = 1.71 \times 10^{-6}$) to encourage smaller weights for more effectve fixed-point quantization.

5.1.4. Spiking Neurons in RNNs: Predictive Maintenance

Building on the approach of integrating spiking mechanisms into recurrent architectures, we applied this concept to the industrial domain of PM [VNK+25]. This application targeted the N_{spikes} term in the energy estimation framework by leveraging the inherent sparsity of spiking neurons within a recurrent architecture designed for resource-constrained edge deployment.

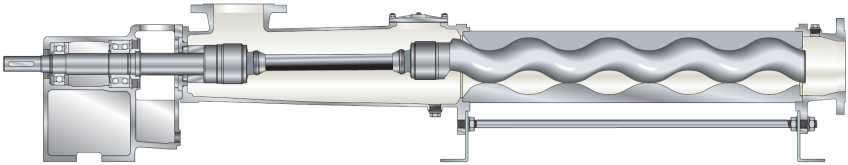


Figure 5.11.: Progressing cavity pump (PCP)

Use Case and Dataset The methodology was applied to a vibration-based monitoring system for a PCP (NETZSCH NM021BY02S [3]), a rotary positive displacement pump used for viscous media and dosing applications (see Figure 5.11). The pump's eccentric and rotating motion creates specific vibration patterns that reflect its operational state. To detect deviations from normal operation, we developed a model that could identify impermissible conditions including overpressure (exceeding 12 bar discharge pressure) and cavitation (suction pressure below 0.75 bar).

Data acquisition was performed using a SIEMENS SITRANS MS200 [6] vibration sensor mounted at the hub position, recording 3-axis accelerometer data at 6664 Hz and logging it via Bluetooth Low Energy (BLE). Additional sensors measured pressure (IFM PG2453 [4]), flow (Endress & Hauser Promag [5]), and pump speed (Omron E2B-M12KN05-M1-B1 [2]). The dataset consisted of approximately 4000 recordings, each containing 16384 data points per axis, totaling about 170 minutes of operation. Each recording was labeled for both regression (flow, pressure, pump speed) and classification (normal, overpressure, cavitation).

Data Preprocessing and Spike Encoding The preprocessing pipeline, illustrated in Figure 5.12, converted raw accelerometer data into spike trains through a multi-stage process:

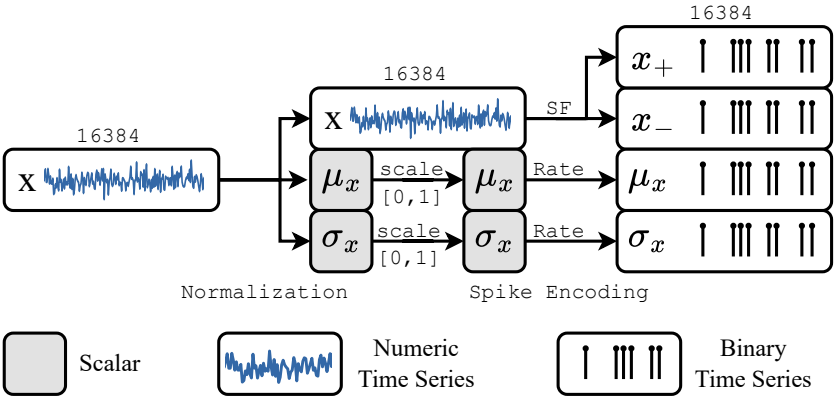


Figure 5.12.: Preprocessing pipeline applied to each accelerometer axis. After global standardization, local normalization produces a normalized time series and extracts local statistics. These components are converted to spike trains using distinct encoding methods: SF encoding for the time series and Poisson rate encoding for the statistical features. Figure adapted from [VNK+25].

1. **Standardization**: Raw data was normalized using global mean and standard deviation calculated from the training dataset. Local statistics (mean and standard deviation) were extracted from each time window to capture segment-specific characteristics and later encoded as additional features.
2. **Spike Encoding**: A hybrid encoding approach converted the processed data into binary spike trains, using the spike encoding tool developed in [VSS+25] and presented in Section 5.1.1. The normalized signal was encoded using the SF algorithm [43], and the local statistics were encoded using Poisson rate encoding, with scalar values normalized to [0,1] and then rate-encoded over the time window.

This preprocessing resulted in 12 input channels for the SNN: for each of the three accelerometer axes, four spike channels were generated (positive

spikes, negative spikes, local mean encoding, and local standard deviation encoding).

Recurrent SNN Architecture The network architecture, shown in Figure 5.13, employed a recurrent SNN designed to process the 12-channel spike train input for simultaneous regression and classification. The architecture consisted of:

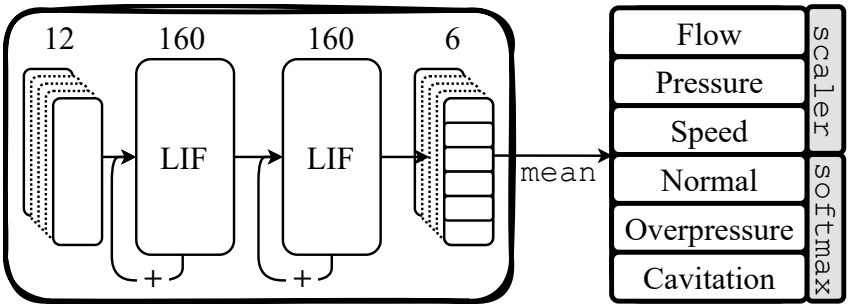


Figure 5.13.: Recurrent SNN architecture for PM. The network processes 12 input spike channels through two recurrent hidden layers with 160 LIF neurons each. Feed-forward and recurrent connections integrate temporal information, with the final layer producing outputs for both regression and classification tasks. Figure adapted from [VNK+25].

1. **Neuron Model**: LIF neurons implemented through the snnTorch library [28], with parameters optimized through hyperparameter search. The membrane potential decay factor $\beta \approx 0.9$ and threshold value of approximately 0.96 determined the spiking dynamics.
2. **Network Structure**: An input projection of size 12, two hidden recurrent LIF layers of size 160, and an output layer of size 6 (3 for regression, 3 for classification). No bias terms were used, and weights were initialized using a normal distribution with mean -0.048 and standard deviation 0.238, values obtained through hyperparameter optimization.
3. **Output Processing**: After sequence processing, outputs from the final linear layer were aggregated by taking their mean across

timesteps. A Softmax activation was applied to the classification outputs and a linear scaler to the regression outputs.

Training Methodology The model was trained using a supervised approach targeting both regression and classification tasks simultaneously:

1. **Data Preparation**: The dataset was split into training (70%), validation (15%), and testing (15%) sets. The training data was augmented using a sliding window approach, extracting segments of 1024 timesteps with a step size of 1, increasing the number of training samples to approximately 40.6 million sequences.
2. **Loss Function**: A composite loss function combined Mean Absolute Error for regression and weighted Cross-Entropy for classification:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{reg}} + 10 \cdot \mathcal{L}_{\text{CE}}^{\text{weighted}} \tag{5.5}$$

   where $\mathcal{L}_{\text{reg}}$ is the sum of absolute errors for the three regression targets, and $\mathcal{L}_{\text{CE}}^{\text{weighted}}$ is the class-weighted cross-entropy loss for the three classification outputs.
3. **Optimization**: The Adam optimizer [46] with a learning rate of 1.224×10^{-2} and weight decay of 1.712×10^{-6} was used. A `ReduceLROnPlateau` scheduler adjusted the learning rate during training, with early stopping implemented with a patience of 10 epochs.
4. **Surrogate Gradient**: Training was enabled using a surrogate gradient function—specifically the fast sigmoid function with a slope parameter of 5—to approximate the derivative of the spiking mechanism during BPTT.

Hyperparameter Optimization The Optuna framework [9] was employed to optimize the hyperparameters, conducting 1000 trials to minimize validation loss. Key optimized parameters included learning rate, LIF neuron parameters (membrane decay, threshold), surrogate gradient slope, weight initialization parameters, and sequence length for training.

Model Compression for Edge Deployment To enable deployment on resource-constrained hardware, we applied compression techniques to the trained SNN:

1. **Pruning**: Magnitude-based weight pruning was applied iteratively, removing the smallest absolute weights in increments of 5%, with a validation loss tolerance threshold of 0.1. This resulted in a weight sparsity of approximately 25%.
2. **Quantization**: Fixed-point quantization followed pruning, with weights quantized to a signed 18-bit format (3 bits for integer, 15 for fractional). Thresholds and decay factors were quantized to an unsigned 16-bit format.

The final compressed model had approximately 80,000 parameters, a memory footprint of 175 KB, and a weight sparsity of 32.82% (increased from pruning due to quantization zeroing out additional small weights). This compression enables the model to run efficiently on neuromorphic hardware, targeting the $N_{\text{connections}}$ term in our energy estimation framework.

5.2. Optimizing Connection Sparsity

Connection sparsity reduction targets the $N_{\text{connections}}$ term by developing architectures where sparse connectivity emerges naturally from spatial constraints rather than requiring post-hoc pruning. We present two approaches: gradient-based optimization of learnable spatial embeddings and gradient-free optimization of fixed spatial arrangements.

5.2.1. Spatial Embedding: Gradient-based Optimization

Building upon the concept of biologically-inspired neural architectures, in [EBV+25] we introduce a parameter-efficient approach where neurons are embedded within Euclidean space and their spatial positions are optimized during training. This method replaces traditional learnable weight matrices with distance-dependent wiring rules, significantly reducing the parameter complexity while maintaining the network's computational capacity.

Spatial Neuron Embedding In this framework, each neuron i is assigned a position $\mathbf{p}_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ in three-dimensional Euclidean space. The connection weight between any two neurons i and j is computed as a differentiable function of their spatial distance:

$$w_{ij} = \frac{1}{\|\mathbf{p}_i - \mathbf{p}_j\|_2} \tag{5.6}$$

This formulation is inspired by the inverse relationship between conductance G and length L in electrical circuits:

$$G = \frac{A}{\rho \cdot L} \tag{5.7}$$

where A represents the cross-sectional area and ρ is the resistivity of the material. This biological analogy reflects how synaptic strength in neural systems often decreases with increasing physical distance between neurons.

Inhibitory Mechanism Since Euclidean distances are inherently non-negative, the distance-based weight computation cannot directly produce negative synaptic weights. To enable both excitatory and inhibitory connections, inspired by biological neural circuits, we introduce a learnable inhibitory mask. Each neuron i is assigned a continuous inhibition value $e_i \in \mathbb{R}$, which is transformed into an inhibition mask E_i using a scaled sigmoid function:

$$E_i = 2 \cdot \sigma_T(e_i) - 1 = 2 \cdot \frac{1}{1 + \exp(-e_i/T)} - 1 \tag{5.8}$$

where $T = 0.1$ controls the steepness of the sigmoid transformation. This formulation ensures that neurons with positive e_i values receive an inhibition mask close to $+1$ (excitatory), while those with negative values receive a mask close to -1 (inhibitory). The inhibition values e_i are initialized uniformly from the range $[-1, 1)$ and optimized during training, allowing the model to learn the optimal distribution of excitatory and inhibitory neurons.

Layer Organization and Constraints To maintain a structured feedforward architecture, neurons are organized into discrete layers by constraining their spatial arrangement. Rather than specifying absolute z-coordinates, we define learnable distances z_l between consecutive layers l and $l + 1$. The cumulative z-coordinates are computed as:

$$z^{(l)} = \sum_{k=1}^{l-1} |z_k| \tag{5.9}$$

This approach ensures proper information flow while allowing neurons to optimize their positions within their respective two-dimensional layers, as illustrated in Figure 5.14. The learnable layer distances enable the network to adapt the spatial organization to the specific requirements of the task.

To prevent the learned layer distances from collapsing to zero during training, we apply a regularization term:

$$\mathcal{L}_{reg} = \lambda \frac{1}{\sum_{l=1}^{L-1} z_l} \tag{5.10}$$

where L is the total number of layers and $\lambda = 0.01$ controls the regularization strength. This term adds a penalty that increases as the sum of layer distances decreases, maintaining meaningful spatial separation between layers.

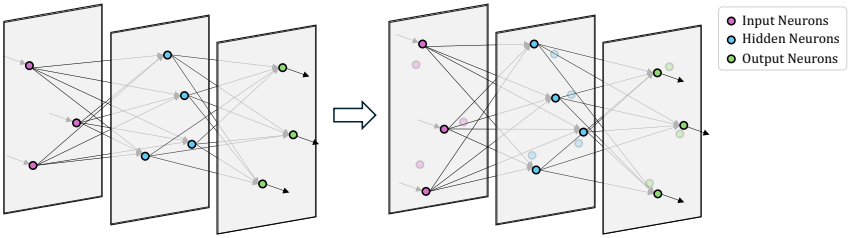


Figure 5.14.: Illustration of a three-layer feedforward network embedded in three-dimensional Euclidean space. Neurons optimize their positions within their respective two-dimensional layers while maintaining fixed z-coordinates corresponding to their layer indices. Figure adapted from [EBV+25].

Parameter Complexity Analysis The spatial embedding approach achieves significant parameter reduction compared to traditional neural networks. For a network with N_L layers, where layer l contains n_l neurons, a conventional Multi-Layer Perceptron (MLP) requires:

$$\sum_{i=1}^{N_L-1} n_i n_{i+1} + \sum_{i=2}^{N_L} n_i \tag{5.11}$$

parameters, accounting for both connection weights and bias terms.

In contrast, a spatially embedded MLP optimizes neuron coordinates, bias terms, and inhibition masks:

$$(d-1) \cdot \sum_{i=1}^{N_L} n_i + \sum_{i=2}^{N_L} n_i + \sum_{i=1}^{N_L-1} n_i \tag{5.12}$$

where $d = 3$ is the embedding dimensionality. Each neuron optimizes only $d - 1 = 2$ coordinates since the z-coordinate remains fixed by layer constraints. This reduction in parameter complexity scales the total parameters from $O(n^2)$ in traditional MLPs to $O(n)$ in spatially embedded networks, where n represents the total number of neurons.

Table 5.1 illustrates the reduced parameter scaling through a concrete example, comparing the parameter counts of a conventional MLP and of an MLP embedded in three-dimensional Euclidean space. Both models consist of 784 input neurons, one hidden layer of 2048 neurons, and 10 output neurons, however, the spatially embedded network achieves a 150-fold reduction in parameters (from 1,628,170 to 10,574).

	MLP	3D MLP
Weights	$784 \cdot 2048 + 2048 \cdot 10 = 1,626,112$	-
Biases	$2048 + 10 = 2,058$	$2048 + 10 = 2,058$
Neuron Coordinates	-	$2 \cdot (784 + 2048 + 10) = 5,684$
Inhibition Mask	-	$784 + 2048 = 2,832$
Total	**1,628,170**	**10,574**

Table 5.1.: Comparison of the parameter counts of a conventional MLP and of an MLP embedded in 3D space with 784 input neurons, one hidden layer with 2048 neurons, and 10 output neurons.

Dataset The MNIST dataset [50] is used for training and evaluation, consisting of 28×28 pixel grayscale images of handwritten digits (0-9). The dataset contains 60,000 training images and 10,000 test images. This results in a network architecture with 784 input neurons corresponding to pixel locations and 10 output neurons for digit classification. Pixel values are normalized to the range [0,1] and training is performed using mini-batches of size 600 with the Adam optimizer [46].

Artificial Neural Network Implementation Spatially embedded MLPs are implemented with neuron positions (x_i, y_i) initialized uniformly from $[0, 1)$, layer distances initialized to 1.0, and bias terms initialized uniformly from $[0, 1)$. Batch normalization is applied after each hidden layer, with Leaky ReLU activation function (negative slope 0.01) and cross-entropy loss. Training employs a learning rate of 0.001 over 300 epochs.

Baselines To ensure meaningful performance comparison, spatially embedded MLPs are evaluated against two conventional MLP baselines. The first baseline uses a single hidden layer with 2048 neurons (1,628,170 parameters), matching the neuron and weight count of spatially embedded networks but with significantly higher parameter complexity. The second baseline employs 14 hidden neurons (11,140 parameters), designed to approximate the parameter count of spatially embedded architectures, but having significantly less neurons and weights. Both baselines use identical training configurations (300 epochs, same initialization and optimization settings) to enable fair comparison.

Pruning Robustness Hypothesis In spatially embedded networks, connection strengths are determined by neuron distances, which leads to a connectivity pattern where strong connections are typically local. We hypothesize that, due to this structure, spatially embedded networks may be more robust to pruning, which is evaluated in the experimental results.

5.2.2. Spatial Embedding: Gradient-free Optimization

For scenarios requiring more flexible architectural exploration, in [VSSB25] we developed evolutionary algorithms that optimize both network topology and neuron parameters within fixed spatial grids.

Motivation and Approach While gradient-based spatial embedding enables optimization of neuron positions and connection weights, it is constrained by the differentiability requirements of backpropagation. EAs offer distinct advantages for SNN optimization, particularly in their ability to directly optimize neuron-specific parameters such as firing thresholds, time constants, and leak potentials—parameters that are challenging to tune effectively through gradient-based methods.

This approach treats spatial embedding as an architectural constraint rather than a learned feature. Neurons are positioned within predefined spatial grids, where connection probabilities and strengths are governed by Euclidean distances between neurons. This framework naturally encourages the emergence of locally connected, sparse networks while allowing optimization of both network topology and individual neuron dynamics.

Spatial Embedding Configurations Four distinct spatial embedding configurations are investigated to understand the impact of dimensionality and spatial arrangement on network performance (Figure 5.15):

- **0D (No Embedding)**: No embedding with uniform connection probability of 0.1
- **1D**: Neurons arranged in a 64 linear configuration
- **2D**: Neurons positioned in an 8×8 square
- **3D**: Neurons distributed in a $4 \times 4 \times 4$ cubic arrangement

These configurations maintain a constant population of 64 hidden neurons while varying spatial dimensionality and arrangement, enabling comparison of spatial embedding effects.

Network Architecture and Neuron Model The recurrent SNN architecture consists of input neurons, a single hidden layer of 64 LIF neurons

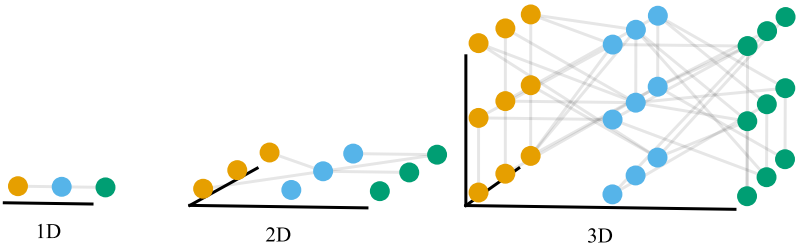


Figure 5.15.: Conceptual visualization of the different spatial embeddings investigated. Figure adapted from [VSSB25].

with sparse recurrent connections, and output neurons, implemented using the Norse framework [76]. Network connectivity is determined by the spatial arrangement of neurons, with connection probability and strength decreasing as a function of Euclidean distance.

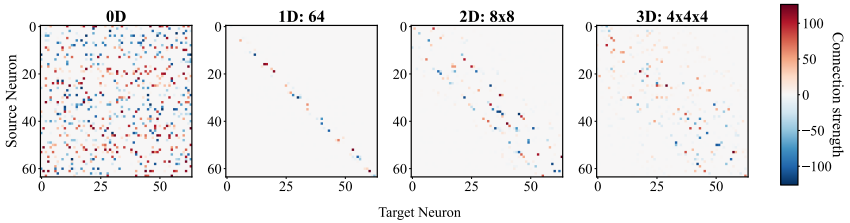


Figure 5.16.: Connectivity strength between hidden neuron pairs for different dimensional embeddings. In 0D, connection probability is uniform. In nD, spatial distance determines connection likelihood and strength. Higher-dimensions allow more connections per neuron. Figure adapted from [VSSB25].

Distance-Dependent Connectivity Connection probability between neurons is determined by their Euclidean distance d_{ij}:

$$P(w_{ij} \neq 0) = \begin{cases} e^{-1}, & \text{if } d_{ij} = 0 \\ e^{-d_{ij}^2}, & \text{otherwise} \end{cases} \tag{5.13}$$

This spatial dependence fosters locally connected networks. A distance-based scaling factor f_{ij}, ranging from 0 to 1, attenuates long-range connection strength, ensuring a high number of near-zero weights (Figure 5.16):

$$f_{ij} = \begin{cases} 1.0, & \text{if } d_{ij} = 0 \\ (e^{-d_{ij}+1})^2, & \text{otherwise} \end{cases} \tag{5.14}$$

Input and output neurons are conceptually placed outside of the hidden recurrent network. The distance from input/output neurons (n_i) to any recurrent layer neuron (n_j) is artificially set to 1, ensuring a connection probability of e^{-1}, and no weight scaling.

Genetic Algorithm Framework The evolutionary optimization employs a genetic algorithm with a population size of 100 individuals evolved over 1000 generations (Figure 5.17). Each individual represents a complete SNN with specific neuron parameters, connection weights, and network topology.

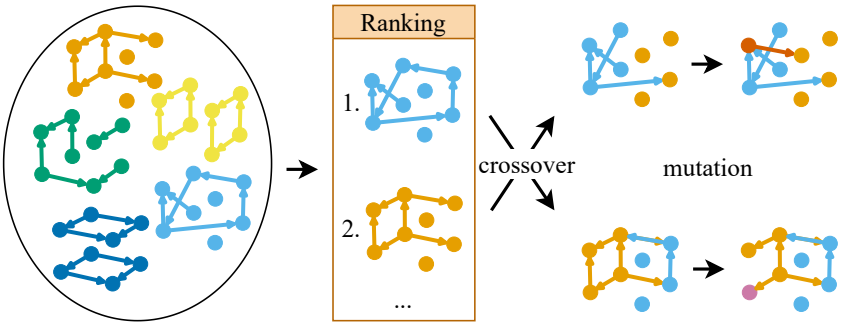


Figure 5.17.: Conceptual visualization of the proposed framework. Neurons are arranged in an n-dimensional grid. The method evaluates a population of networks and ranks them. High-ranking genes produce modified duplicates through crossover and mutation. Figure adapted from [VSSB25].

Fitness Evaluation Each SNN is evaluated on continuous control tasks using three training seeds and 100 testing seeds. For training and evaluation, we used the Gymnasium library [94], which provides standard RL environments like MuJoCo [93], ideal for testing control algorithms. The primary fitness metric is the mean reward across training seeds, supplemented by behavioral descriptors that capture network characteristics:

- f_1: Number of active output neurons (quantifying joint usage)
- f_2: Standard deviation of active output neuron values over time (determining movement amount)
- f_3: Connection sparsity between synapses (measuring solution efficiency)

The final fitness score combines these metrics using weighted importance: reward (100), sparsity (10), joint usage (1), and movement variability (0.1).

Selection Strategy Elite selection identifies $m = \lfloor \sqrt{n_p} \rfloor = 10$ top-performing networks based on the weighted fitness score. The next generation comprises 90% from elite-based reproduction (10 elites, 10 mutated elites, and 70 offspring from elite crossover) and 10% randomly initialized networks.

Variation Operators Genetic diversity is introduced through variation operators:

Crossover: Two parent networks select half of their neurons at random and exchange them along with their parameters and outgoing connections. This operation preserves the spatial structure while combining successful network motifs from different individuals.

Mutation: Stochastic modifications [operation (probability)]:

- Synaptic connections: Addition (0.01), deletion (0.05), or weight modification (0.1)
- Neuron properties: Polarity alteration (0.05), threshold adjustment (0.1), reset potential modification (0.1), leak potential adjustment (0.1), membrane time constant modification (0.1), and synaptic time constant adjustment (0.1)

Network Cleanup After variation operations, networks undergo structural validation to ensure functional connectivity. Neurons lacking input connections (excluding self-connections), neurons without output connections, and isolated subnetworks are automatically removed.

Input/Output Processing Environmental observations are normalized to the range [0, 1] based on empirically determined bounds through random sampling. These normalized values are multiplied by input connection weights and fed to the hidden neuron layer.

Network outputs are decoded by integrating the dendritic currents of output neurons over multiple timesteps. The aggregated currents are normalized using a sigmoid function and scaled to match the action space bounds of the control environment.

The network processes each environmental step over a fixed number of timesteps, allowing temporal integration of information. Neuron states persist across environment steps within an episode, preserving memory and temporal dynamics, but reset at episode boundaries.

Hypotheses and Expected Benefits This gradient-free spatial embedding approach is hypothesized to offer several advantages over traditional Neural Architecture Search (NAS) and gradient-based methods. We also aim to investigate whether spatially embedded networks can provide evidence for cortical column-like organization, with the expectation that a two-dimensional (2D) spatial embedding will be the most effective among the different dimensionalities.

1. **Sparse Connectivity**: Distance-dependent connections promote sparse, locally connected networks that not only achieve high performance with fewer parameters, but also exhibit increased robustness to pruning.
2. **Biological Plausibility and Cortical Columns**: Spatial constraints mirror biological neural organization, potentially leading to the emergence of robust and interpretable network motifs. By comparing different spatial embeddings (e.g., 1D, 2D, 3D), we seek evidence that 2D organization will be most effective, mirroring the observation of cortical columns in biological brains.

3. **Optimization Flexibility**: Direct parameter evolution enables tuning of neuron dynamics and properties beyond just connection weights.

6. Validation of Sparsity-Inducing Methods in SNNs

This chapter validates the proposed energy minimization methods for SNNs, focusing on reducing N_{spikes} and $N_{\text{connections}}$ in Equation 3.10. The section also presents validation of the applied methods through real-world use cases spanning biomedical signal processing, predictive maintenance, brain-machine interfaces, and control systems.

6.1. Optimizing Activation Sparsity

This section evaluates methods for reducing the N_{spikes} term through optimized spike encoding and integration of spiking neurons within ANN architectures.

6.1.1. A Framework for Optimized Spike Encoding

Following the framework development described in Section 5.1.1, I evaluated the four temporal encoding methods (LIF, SF, PWM, and BSA) to assess their effectiveness in minimizing N_{spikes} while preserving information fidelity. The evaluation focused on reconstruction accuracy, spike sparsity, and energy efficiency on embedded hardware.

6.1.1.1. Performance

Table 6.1 presents the reconstruction error (MSE) for each encoding method across the four signal types defined in the methods. SF encoding achieved the lowest overall reconstruction error (mean MSE of 0.1614), demonstrating particular effectiveness with sinusoidal and trended signals (MSE of 0.000139 and 0.000970 respectively). BSA showed superior performance for rectangular signals (MSE of 0.0636). LIF encoding performed best with vibration signals (MSE of 0.370641), while PWM exhibited the highest overall reconstruction error (mean MSE of 0.3227).

Table 6.1.: Reconstruction Error (MSE) of different Spike Encoding Algorithms by Signal Type

Signal Type	LIF	SF	PWM	BSA
Vibration	**0.370641**	0.487395	1.099153	0.845590
Trended	0.102157	**0.000970**	0.015004	0.006510
Rectangular	0.081864	0.157202	0.172042	**0.063650**
Sinusoidal	0.126749	**0.000139**	0.004631	0.013472
Mean Error	0.170353	**0.161427**	0.322707	0.232305

6.1.1.2. Sparsity Analysis

Table 6.2 quantifies the spike sparsity achieved by each encoding method, directly relating to the N_{spikes} term in Equation 3.10. PWM encoding achieved the highest sparsity (17.75% mean spike rate), particularly effective for signals with smooth temporal dynamics such as trended (3.66%)

and sinusoidal (3.02%) signals. SF encoding provided moderate sparsity (30.60% mean) while maintaining good reconstruction accuracy across all signal types. LIF and BSA exhibited higher spike rates (51.90% and 53.08% respectively), reducing their potential energy efficiency benefits despite their reconstruction performance.

Table 6.2.: Spike Sparsity of different Spike Encoding Algorithms by Signal Type (% of total timesteps)

Signal Type	LIF (%)	SF (%)	PWM (%)	BSA (%)
Vibration	**63.17**	58.74	50.00	46.19
Trended	34.41	64.18	**96.34**	38.66
Rectangular	37.53	**89.80**	85.67	51.52
Sinusoidal	57.28	64.86	**96.98**	51.30
Mean Sparsity	48.10	69.40	**82.25**	46.92

6.1.1.3. Energy Evaluation

The C++ implementations tested on the MCX-N947-EVK development board revealed significant differences in computational requirements. Table 6.3 presents the practical energy implications of each encoding method, with SF serving as the baseline due to its superior overall performance.

SF encoding demonstrated the highest energy efficiency with the shortest encoding time (123.52 ms) and lowest dynamic power consumption (6.87 nW). LIF encoding showed comparable performance with only 41.67% additional energy consumption, making it a viable alternative. PWM required substantially more energy (812.50% relative to SF) due to its complex carrier signal operations, while BSA exhibited the highest computational cost (5083.33% relative energy) due to its iterative FIR filter computations.

6.1.1.4. Discussion

These results demonstrate the trade-offs between spike sparsity and reconstruction accuracy for different encoding methods. The parameter

Table 6.3.: Energy Efficiency of Spike Encoding Algorithms on Embedded Hardware

Metric	LIF	SF	PWM	BSA
Encoding Time (ms)	127.15	**123.52**	691.32	2238.58
Dynamic Power (nW)	9.71	**6.87**	11.41	20.00
Dynamic Energy (pWh)	0.34	**0.24**	2.19	12.44
Time vs. SF (%)	2.94	**0.00**	459.64	1712.28
Power vs. SF (%)	41.34	**0.00**	66.08	191.12
Energy vs. SF (%)	41.67	**0.00**	812.50	5083.33

optimization focused on minimizing reconstruction error rather than co-optimizing for sparsity, prioritizing information preservation over sparsity. PWM achieved the highest sparsity (17.75% mean spike rate, reduction of 5.6× compared to a dense input) but required higher computational cost, while SF balanced reconstruction accuracy and energy efficiency.

These encoding methods target the N_{spikes} term in the energy minimization framework (Equation 3.10) by transforming continuous signals into sparse spike trains. This conversion achieves up to 5.6× reduction in input layer neuron activity while maintaining information fidelity, demonstrating the potential for power-efficient SNN operation even without explicit spike rate minimization.

6.1.2. Spiking Neurons in CNNs: Sleep Analysis

A key strategy for reducing neuron activation is the replacement of ANN activation functions with spiking neurons. This creates hybrid models that benefit from sparse, event-driven updates by combining the feature extraction capabilities of ANNs with the efficiency of spiking neurons.

This subsection validates this approach by integrating LIF neurons into convolutional architectures using the sleep stage classification architecture described in Section 5.1.2, which was presented in [BVH+24].

6.1.2.1. Performance

The SNN and ANN models were evaluated on binary, three-class, and five-class sleep staging tasks, in addition to an obstructive sleep apnea detection task. The results are presented using precision, recall, F1-Score, Area Under the Curve (AUC), accuracy, and Cohen's Kappa.

For binary sleep-wake classification, both models achieved high performance, as shown in Table 6.4. The SNN yielded a higher recall (0.9462) and accuracy (0.9469), while the ANN achieved better precision (0.9770) and a higher F1-score (0.9451).

Table 6.4.: SNN and ANN Performance for Binary Sleep-Wake Classification

	SNN	**ANN**
Precision	0.9109	**0.9770**
Recall	**0.9462**	0.9153
F1	0.9268	**0.9451**
AUC	0.9462	**0.9745**
Accuracy	**0.9469**	0.9169
κ	**0.8538**	0.7751

κ Cohen's Kappa

In the three-class classification task (Wake-NREM-REM), the SNN model demonstrated superior overall performance compared to the ANN, achiev-

ing a higher Macro-F1 score (0.8161 vs. 0.7850) and accuracy (0.8496 vs. 0.8213), as detailed in Table 6.5.

Table 6.5.: SNN and ANN Classification Performance for Three-Class (Wake-NREM-REM) Sleep Staging

Model	Scores	Wake	NREM	REM
SNN	**Precision**	**0.7764**	**0.9577**	**0.6105**
	Recall	**0.8915**	**0.8234**	**0.9163**
	F1-Score	**0.8300**	**0.8855**	**0.7327**
	AUC	0.9385	**0.9234**	0.9482
	Accuracy		**0.8496**	
	κ		**0.7258**	
	F1$^{\mathrm{W}}$		**0.8550**	
	F1$^{\mathrm{M}}$		**0.8161**	
ANN	**Precision**	0.7507	0.9429	0.5591
	Recall	0.8910	0.7891	0.8707
	F1	0.8149	0.8591	0.6810
	AUC	**0.9523**	0.9140	**0.9618**
	Accuracy		0.8213	
	κ		0.6782	
	F1$^{\mathrm{W}}$		0.8281	
	F1$^{\mathrm{M}}$		0.7850	

κ Cohen's Kappa, F1$^{\mathrm{W}}$ (Weighted), F1$^{\mathrm{M}}$ (Macro)

For the five-class classification task, the SNN again outperformed the ANN across most metrics, achieving a Macro-F1 score of 0.6843 compared to the ANN's 0.6272 (see Table 6.6).

In the detection of OA and OH events, the SNN model also showed superior performance over the ANN, with a Macro-F1 score of 0.6832 compared to 0.4930 (Table 6.7). The performance was likely constrained by the low prevalence of apnea events (5%) in the dataset.

A comparative summary of these results against other state-of-the-art methods is provided in Table A.2. Our models establish new state-of-the-

Table 6.6.: SNN and ANN Classification Performance for Five-Class (Wake-N1-N2-N3-REM) Sleep Staging

Model	Scores	Wake	N1	N2	N3	REM
SNN	**Precision**	**0.8839**	**0.4345**	0.6991	**0.7645**	**0.6363**
	Recall	0.7532	**0.6243**	**0.6212**	0.6993	**0.7972**
	F1	0.8133	**0.5124**	**0.6578**	**0.7305**	**0.7077**
	AUC	0.9395	0.8098	0.8224	0.9015	0.9187
	Accuracy			**0.6877**		
	κ			**0.6019**		
	F1^{W}			**0.6943**		
	F1^{M}			**0.6843**		
ANN	**Precision**	0.8149	0.3881	**0.7164**	0.6185	0.5800
	Recall	**0.8319**	0.4236	0.4147	**0.8424**	0.7907
	F1	**0.8233**	0.4051	0.5253	0.7133	0.6691
	AUC	**0.9628**	**0.8125**	**0.8283**	**0.9319**	**0.9486**
	Accuracy			0.6456		
	κ			0.5852		
	F1^{W}			0.6354		
	F1^{M}			0.6272		

κ Cohen's Kappa, F1^{W} (Weighted), F1^{M} (Macro)

art Macro-F1 scores for binary (0.9451), three-class (0.8161), and five-class (0.6843) sleep stage classification from single-lead ECG.

6.1.2.2. Sparsity Analysis

Analysis of the LIF layer activations across 67,200 samples revealed significant sparsity in the convolutional layers (see Figure 6.1). Table 6.8 presents the detailed sparsity characteristics for each LIF layer. The three LIF layers achieved average sparsity levels of 85.51%, 80.02%, and 78.62% respectively, with a weighted average sparsity of 84.25% across all layers. This corresponds to approximately 6× reduction in synaptic activations compared to a dense ANN during a single forward pass.

Table 6.7.: Per-segment ANN and SNN Classification Performance for Obstructive Apnea (OA) and Obstructive Hypopnea (OH)

Model	Scores	None	OA	OH
SNN	**Precision**	**0.9927**	**0.4668**	**0.3505**
	Recall	**0.8981**	**0.8797**	**0.8518**
	F1	**0.9430**	**0.6100**	**0.4966**
	AUC	**0.9349**	0.9369	**0.9156**
	Accuracy		**0.8955**	
	κ		**0.5108**	
	F1$^{\mathrm{W}}$		**0.9140**	
	F1$^{\mathrm{M}}$		**0.6832**	
ANN	**Precision**	0.9859	0.1626	0.1951
	Recall	0.8881	0.5686	0.5768
	F1	0.9345	0.2529	0.2916
	AUC	0.9128	**0.9548**	0.8920
	Accuracy		0.8739	
	κ		0.2805	
	F1$^{\mathrm{W}}$		0.9049	
	F1$^{\mathrm{M}}$		0.4930	

κ Cohen's Kappa, F1$^{\mathrm{W}}$ (Weighted), F1$^{\mathrm{M}}$ (Macro)

6.1.2.3. Discussion

The SNN architecture achieved performance superior or comparable to its ANN counterpart across all sleep stage classification tasks, with particular advantages in complex three- and five-class scenarios (Tables 6.5 and 6.6). Our models established new state-of-the-art results for single-lead ECG-based classification (Table A.2), suggesting that LIF neurons effectively capture temporal features in ECG signals.

The SNN also exhibited significant activation sparsity, achieving 84.25% sparsity across convolutional layers (Table 6.8). This corresponds to a 6-fold reduction in synaptic operations (N_{spikes}) per forward pass compared to dense networks, providing a key advantage for energy-efficient implementations.

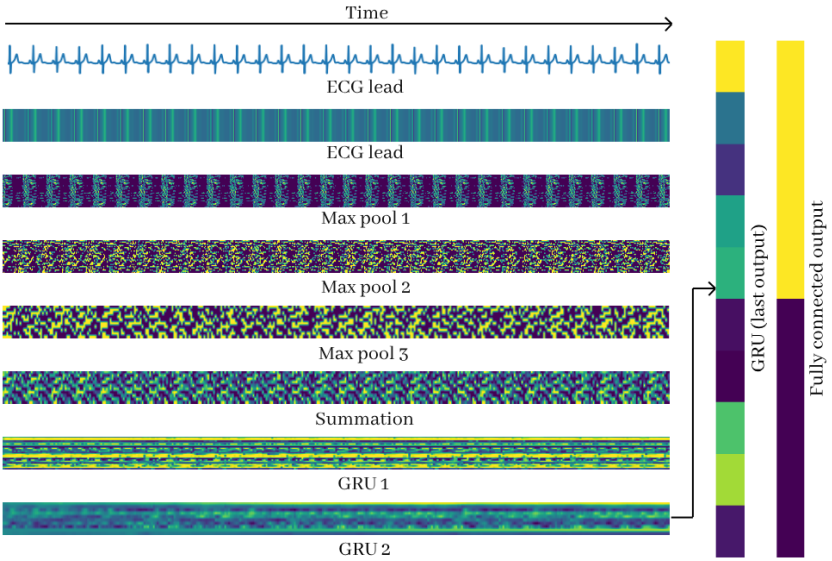


Figure 6.1.: Activation patterns showing spiking behavior across different network layers during ECG-based sleep classification. Figure adapted from [BVH+24].

However, the temporal repetition strategy creates a trade-off. While the 10 repetitions enable high classification accuracy, they increase the total computational load, resulting in synaptic activations that exceed a single ANN forward pass by approximately 1.6×. Reducing repetitions to fewer than six would make the SNN computationally more efficient than the ANN, affecting the N_{spikes} term in the energy estimation framework. Since the observed sparsity was achieved without sparsity-aware training, further computational and energy reductions are possible.

Table 6.8.: LIF Layer Activation Sparsity in Convolutional Architecture

Layer	Neurons	Spikes/Sample	Spike Rate (%)	Sparsity (%)
LIF1	178,560	25,867 ± 4,589	**14.49** ± 2.57	**85.51** ± 2.57
LIF2	43,770	8,744 ± 471	**19.98** ± 1.08	**80.02** ± 1.08
LIF3	7,100	1,518 ± 126	**21.38** ± 1.77	**78.62** ± 1.77
Total	229,430	36,129 ± 5,186	**15.75** ± 2.01	**84.25** ± 2.01

6.1.3. Spiking Neurons in RNNs: Neural Decoding

This subsection evaluates the integration of spiking recurrent units into hybrid neural decoding architectures for BMI applications. We replace conventional GRU units with spiking alternatives (sGRU and LIF) to achieve sparse, event-driven temporal processing. The evaluation uses the neural decoding architectures described in Section 5.1.3, which were presented in [VKKB24] and [KVKB25].

6.1.3.1. Performance

The performance of the neural decoding architectures from Section 5.1.3 was evaluated on the Primate Reaching Dataset.

The results for all six model variants presented in [VKKB24] are compared against the NeuroBench baselines [108] in Table 6.9.

Table 6.9.: Performance comparison of the proposed neural decoding models and the NeuroBench baselines. Models prefixed with 'B-' are baselines from [108]. Model architectures are described in Section 5.1.3. Metrics are averaged over all test samples.

Track	Model	Footprint	Dense	MACs	ACs	R^2
	B-ANN3	137752	33888	11507	0	0.615
	B-SNN3	**33996**	43680	32256	5831	0.633
Track 1	GRU-t1	352904	22342	8518	793	**0.707 ± 0.012**
	sGRU-t1	425924	22318	7238	797.7	0.656 ± 0.013
	LIF-t1	302492	**20766**	6414	825	0.648 ± 0.022
	B-ANN2	**27160**	6237	4970	0	0.576
	B-SNN2	29248	7300	0	414	0.581
Track 2	GRU-t2	174104	4947	627	248	**0.621 ± 0.014**
	sGRU-t2	180716	4932	379	250.2	0.577 ± 0.013
	LIF-t2	168596	**4631**	201	254	0.566 ± 0.016

In the accuracy-focused Track 1, all proposed models outperform their respective baselines. The GRU-t1 model achieves the highest R^2 score of 0.707, improving upon the B-ANN3 baseline's 0.615 while requiring

26% fewer MACs and 34% fewer dense operations. In Track 2, which emphasizes efficiency, the models demonstrate significant computational savings. The GRU-t2 model improves the R^2 score to 0.621 (from 0.576 for B-ANN2) with only 13% of the MACs.

A comparison of the recurrent units reveals clear performance trade-offs. GRU-based models consistently yield the highest R^2 scores. In contrast, LIF-based models are the most computationally efficient (Dense, MACs) but at the cost of a slightly lower R^2 score. The sGRU models provide a balance, achieving higher R^2 scores than the LIF models, which suggests their gating mechanism helps preserve temporal information.

Figure A.1 provides a qualitative view of the decoding performance, showing the predicted velocity outputs of the three Track 2 model types for exemplary test samples. The figure illustrates cases of high-fidelity reconstruction ($R^2 \approx 0.9$), average performance ($R^2 \approx 0.7$), and poor reconstruction ($R^2 \ll 0$), offering insight into the models' behavior on different movement dynamics.

To further analyze performance, the influence of the number of predicted keypoints was investigated. As detailed in Table 6.10, reducing the number of keypoints for the GRU model—and thereby increasing the interpolation step size—consistently improves the R^2 score. This finding validates the chosen interpolation-based architecture, which enhances decoding accuracy while simultaneously reducing the computational load.

Table 6.10.: Comparison of R^2 scores for a GRU-based model with a varying number of keypoints, trained on a subset of the data.

Keypoints (Interpolation)	Dense Ops	Effective MACs	R^2
1025 (1-step)	46400.3	37184.3	0.736
513 (2-step)	27808.3	18592.3	0.766
257 (4-step)	20054.3	10838.3	0.764
129 (8-step)	17734.3	8518.3	0.779

Further development in [KVKB25] focused on optimizing the architecture for real-time performance, leading to the creation of three new model variants: BMnet, RTnet, and sRTnet.

BMnet, designed to maximize accuracy, surpasses previous benchmarks for both ANN and SNN models on the Primate Reaching task, achieving an R^2 score of 0.717, as shown in Table 6.11. RTnet was developed to balance high accuracy with real-time constraints, achieving an R^2 score of 0.685.

Table 6.11.: Performance comparison of the high-accuracy models from [KVKB25] (BMnet, RTnet) against other state-of-the-art models. The models from this work achieve a new state-of-the-art in decoding accuracy.

Model	**Event-Based**	**Test R^2**
SNN3 [108]	✓	0.633
bigRSNN [56]	✓	0.698 ± 0.002
LIF-t1 (ours)	✓	0.648 ± 0.022
ANN3 [108]	✗	0.615
GRU-t1 (ours)	✗	0.707 ± 0.012
Wang et al. [102]	✗	0.710 ± 0.050
BMnet (ours)	✓	$\mathbf{0.717 \pm 0.004}$
RTnet (ours)	✓	0.685 ± 0.006

The sRTnet model, a compressed version of RTnet, was designed for resource-constrained scenarios. As detailed in Table 6.12, sRTnet achieves the highest R^2 score (0.675) among the efficiency-focused models while remaining real-time capable. However, this performance comes at the cost of an increased memory footprint and a higher number of MACs compared to the tinyRSNN model from [56], which is a trade-off dependent on the specific hardware constraints of the target deployment platform.

6.1.3.2. Sparsity Analysis

To evaluate activation sparsity in spiking recurrent layers, we analyzed the output activation patterns of GRU, sGRU, and LIF units using the neural decoding architecture described in Section 5.1.3. The comparison was conducted on the test set of the Primate Reaching Dataset with track 1 (75776 samples, 64 recurrent units) and track 2 (151552 samples, 20 recurrent units) models from [VKKB24], with activation patterns visualized

Table 6.12.: Performance comparison of the compressed, real-time capable sRTnet model with other efficiency-focused models. sRTnet achieves a new state-of-the-art R^2 score among real-time capable, event-based models. EB: Event-Based, RT: Real-time Capable.

Model	EB	Test R^2	Mem. (kB)	MACs	ACs	RT
B-SNN2 [108]	✓	0.581	29	0	414	✓
tinyRSNN [56]	✓	0.660 ± 0.002	**27**	0	304	✓
LIF-t2 (ours)	✓	0.566 ± 0.016	169	201	254	✗
B-ANN2 [108]	✗	0.576	27	4970	0	✗
GRU-t2 (ours)	✗	0.621 ± 0.014	174	627	248	✗
sRTnet (ours)	✓	$\mathbf{0.675 \pm 0.011}$	105	12274	326	✓

in Figure 6.2. Table 6.13 summarizes the measured activation sparsity for each model type and track. LIF-based recurrent layers consistently achieve the highest activation sparsity (around 94% in both model sizes), followed by sGRU with intermediate sparsity levels (65-69%), while conventional GRU units exhibit dense activations.

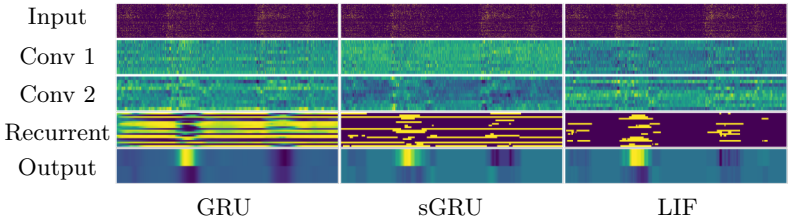


Figure 6.2.: Recurrent layer activation patterns comparing GRU (dense), sGRU (moderate sparsity), and LIF (high sparsity) units, demonstrating the progression toward sparse, energy-efficient processing while maintaining temporal modeling capabilities. Figure adapted from [VKKB24].

For real-time neural decoding, we evaluated three model variants (BMnet, RTnet, sRTnet) using two recurrent LIF layers of size 56, as described in Section 5.1.3 and introduced in [KVKB25]. Activation sparsity was measured across both layers for all test samples (151552 samples for sRTnet,

Table 6.13.: Activation sparsity of recurrent layers in neural decoding models for both challenge tracks (Track 1: 64 neurons, 75776 samples; Track 2: 20 neurons, 151552 samples). Values are mean ± standard deviation over test samples.

Model	Track 1	Track 2
GRU	0	0
sGRU	0.651 ± 0.017	0.69 ± 0.07
LIF	0.939 ± 0.008	0.946 ± 0.009

RTnet and 75776 samples for BMnet). Table 6.14 summarizes the results. Both RTnet and sRTnet achieve high activation sparsity (≈86%), while BMnet shows slightly lower sparsity (≈82%).

Table 6.14.: Activation sparsity of recurrent layers in real-time neural decoding models. Values are mean ± standard deviation over all test samples.

Model	Activation Sparsity
sRTnet	0.859 ± 0.002
RTnet	0.858 ± 0.001
BMnet	0.818 ± 0.001

6.1.3.3. Discussion

The results demonstrate a clear trade-off between decoding accuracy and computational efficiency. In the initial models from [VKKB24], conventional GRU units delivered the highest R^2 scores but with dense activations. The introduction of spiking units led to significant efficiency gains: LIF-based models achieved up to 94% activation sparsity, suggesting a potential 20-fold reduction in synaptic operations compared to GRUs, albeit with a slight dip in accuracy. sGRU models offered a middle ground, retaining strong performance with moderate sparsity (around 67%).

The real-time models from [KVKB25] further refined this balance. BMnet set a new state-of-the-art accuracy (R^2 of 0.717), while sRTnet achieved the best performance (R^2 of 0.675) among real-time capable, event-based models. This performance came with a trade-off, as sRTnet required a larger memory footprint than competitors like tinyRSNN. These models also maintained high sparsity (82-86%), achieving a 6-7× reduction in synaptic activations.

Overall, the findings validate integrating spiking neurons into recurrent architectures for neural decoding. The developed models advance the state-of-the-art in performance and demonstrate a path toward hardware-efficient, real-time brain-computer interfaces by leveraging event-driven computation.

6.1.4. Spiking Neurons in RNNs: Predictive Maintenance

Building on the previous applications, this section evaluates the recurrent SNN architecture in the context of an industrial PM task. This use case, based on the work in [VNK+25] and detailed in Section 5.1.4, serves as an example of applying the energy minimization strategies from Chapter 3 to a complex, real-world problem. The evaluation focuses on task performance, the resulting activation and connection sparsity, and the estimated energy consumption to assess the model's suitability for edge deployment.

6.1.4.1. Performance

The performance of the trained and compressed SNN model was evaluated on the test set, comprising 606 samples. The evaluation focused on the model's ability to perform simultaneous multi-task regression and classification, as detailed in Section 5.1.4.

The model's classification performance for normal, overpressure, and cavitation conditions is summarized in Table 6.15 and visualized in Figure 6.3. The overall accuracy is high, exceeding 97%. A critical result is the False Negative Rate (FNR) of 0.0% for both overpressure and cavitation, indicating that the model correctly identified all critical fault conditions in the test set. The False Positive Rate (FPR) for overpressure was 3.9%, meaning a small fraction of normal samples were incorrectly flagged.

Table 6.15.: Model Classification Performance on the Test Set.

Metric	Normal	Overpressure	Cavitation	Overall
Accuracy	0.9692	0.9692	1.0000	0.9795
F1 Score	0.9758	0.9339	1.0000	0.9699
AUC	0.9891	0.9904	1.0000	0.9932
FPR	0.0000	0.0394	0.0000	0.0131
FNR	0.0472	0.0000	0.0000	0.0157

For the regression task, the model predicted flow (m^3/h), pressure (bar), and pump speed (min^{-1}). Figure 6.4 illustrates the raw and smoothed model outputs against the target values. As summarized in Table 6.16,

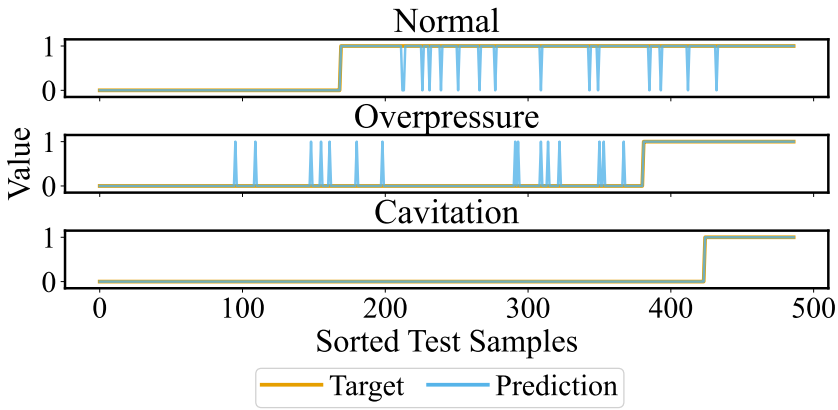


Figure 6.3.: Classification performance on the sorted test set. Targets (orange) and predictions (light blue) are shown. Figure adapted from [VNK+25].

applying a moving median filter (window size 10) to the raw output significantly improved all performance metrics.

The smoothed predictions for flow achieved an Mean Relative Percentage Error (MRPE) of 0.93%, which is within the typical range for industrial sensors. For pump speed, the Mean Absolute Percentage Error (MAPE) of 3.12% and MRPE of 0.93% are close to industrial standards. The regression performance for pressure (MRPE of 2.46%) was less accurate and could be a subject for future improvements. The high MAPE values for pressure are influenced by target values near zero, making MRPE a more reliable metric in this case.

6.1.4.2. Sparsity Analysis

The model consists of two recurrent hidden layers with 160 LIF neurons each with an input layer of size 12 (332 neurons in total). During inference on the test set (606 sequences of 16384 timesteps), the network generated on average $530{,}317 \pm 73{,}104$ spikes per sample (Table 6.17). This corresponds to approximately 32.4 ± 4.5 spikes per timestep across all neurons, or a

Table 6.16.: Model Regression Performance on the Test Set. Industry benchmarks for Measured Value Error (MVE) and Full Scale Error (FSE) are provided for comparison.

Metric	Flow (m^3/h)	Pressure (bar)	Speed (min^{-1})
MAE Raw	0.021	0.63	10.50
MAE Smooth	0.011	0.40	6.90
MAPE Raw (%)	15.37	9237.21	4.69
MAPE Smooth (%)	4.97	2447.47	3.12
Industry MVE (%)	–	–	0.1–2.0
MRPE Raw (%)	1.78	3.87	1.41
MRPE Smooth (%)	0.93	2.46	0.93
Industry FSE (%)	0.1–5.0	0.15–1.0	–

per-neuron spiking probability of 0.0975 ± 0.0134. Thus, each LIF neuron spikes in $9.75\% \pm 1.34\%$ of timesteps on average, yielding an activation sparsity of $90.25\% \pm 1.34\%$. This sparsity was achieved without explicit spike regularization in the loss function, relying on the inherent dynamics of the LIF neuron.

Table 6.17.: Spike generation by network layer for the PM SNN model (mean ± std over 606 test samples).

Layer	Neurons	Spikes	Spike Rate
input	12	$59{,}121 \pm 6{,}825$	0.3007 ± 0.0347
lif0	160	$195{,}677 \pm 33{,}864$	0.0746 ± 0.0129
lif1	160	$275{,}519 \pm 66{,}654$	0.1051 ± 0.0254
TOTAL	332	$530{,}317 \pm 73{,}104$	0.0975 ± 0.0134

After training, the model was compressed using magnitude-based weight pruning and fixed-point quantization. The final model exhibited a weight sparsity of 32.82%, meaning about one-third of the connections were zeroed out. This reduction in non-zero connections lowers the $N_{\text{connections}}$ term in

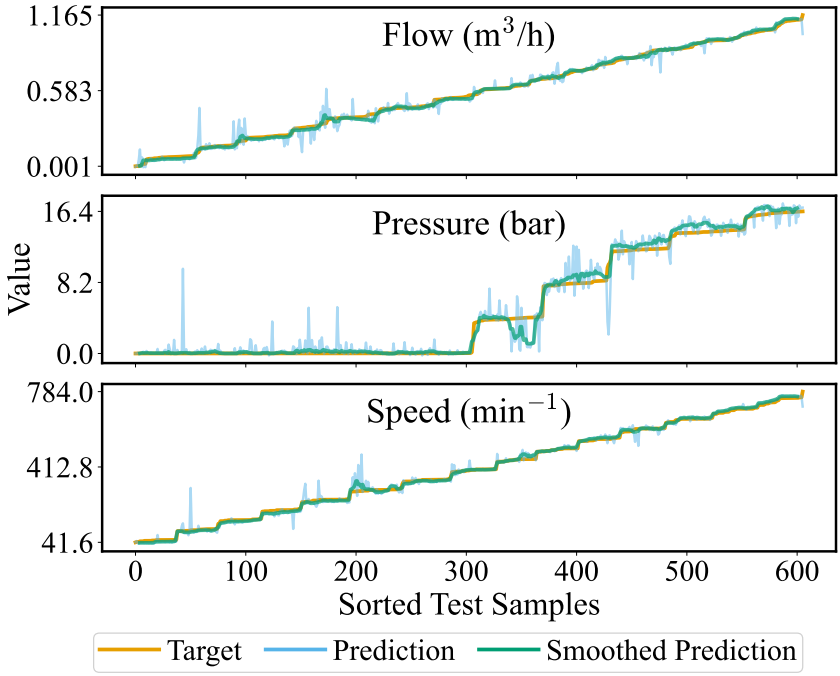


Figure 6.4.: Regression performance on the sorted test set for flow (m^3/h), pressure (bar), and pump speed (min^{-1}). Targets (orange), raw predictions (light blue), and smoothed predictions (moving median filter, window size 10, green) are shown. Figure adapted from [VNK+25].

the energy estimation framework. The total number of spiking connections after pruning was 53,527 (from 79,680 in the dense model).

6.1.4.3. Energy Evaluation

Energy consumption per inference was estimated using the methodology in Chapter 3, with hardware-specific energy costs from Table A.1. Table 6.18 summarizes the estimated energy per inference for the compressed SNN

model on different hardware platforms, with standard deviations across all test samples.

Table 6.18.: Estimated Energy per Inference (J) for the PM SNN model (mean ± std over 606 samples).

Device	Energy (J)		
	Total	Synaptic	Neuron
x86	1.13×10^{1}	1.12×10^{1}	4.59×10^{-2}
ARM	1.18×10^{0}	1.17×10^{0}	4.81×10^{-3}
Loihi	$\mathbf{(3.16 \pm 0.33) \times 10^{-3}}$	$\mathbf{(2.73 \pm 0.33) \times 10^{-3}}$	$\mathbf{4.33 \times 10^{-4}}$

The neuromorphic platform (Loihi) achieves the lowest estimated energy consumption per inference (3.16 ± 0.33 mJ), with synaptic operations accounting for about 86% of the total. This low synaptic energy footprint results from the combined effect of:

1. **Activation sparsity:** $\approx 90\%$ in the recurrent layers, corresponding to a $\approx 10\times$ reduction in N_{spikes}.
2. **Connection sparsity:** $\approx 33\%$ ($\approx 1.5\times$ reduction in $N_{\text{connections}}$).
3. **Hardware efficiency:** Loihi provides about $300\times$ lower synaptic energy per operation compared to x86 and $30\times$ compared to ARM.

Since synaptic energy is proportional to the product $N_{\text{spikes}} \times N_{\text{connections}}$, the estimated synaptic energy reduction is approximately $10 \times 1.5 \times 300 \approx 4500\times$ compared to x86 and $10 \times 1.5 \times 30 \approx 450\times$ compared to ARM, with $15\times$ achieved solely through software-level improvements. Further reductions in spike rates or increases in connection sparsity could yield additional energy savings.

6.1.4.4. Discussion

The results validate the software-level energy minimization concept outlined in Chapter 3. The SNN demonstrated reliable performance on a complex industrial task, achieving high classification accuracy—with zero FNR for critical faults—and robust regression for most targets.

The energy efficiency stems from reducing both software-level terms in the energy estimation framework (Equation 3.10). The model's recurrent LIF layers reached an activation sparsity (N_{spikes}) of ≈90%, a 10× reduction in spike-driven operations. At the same time, pruning and quantization produced a connection sparsity ($N_{\text{connections}}$) of ≈33%, a 1.5× reduction in synaptic connections.

Together, these software-level improvements yield an estimated 15× reduction in synaptic energy. Paired with the hardware efficiency of a neuromorphic platform like Loihi, the total estimated energy savings increase up to 4500× over an x86 CPU. This underscores the potential of a combined software-hardware approach for creating efficient, on-device Predictive Maintenance (PM) solutions.

6.2. Optimizing Connection Sparsity

This section evaluates methods for reducing the $N_{\text{connections}}$ term in Equation 3.10 through spatial embedding approaches. We investigate two spatial embedding methodologies: gradient-based optimization where neuron positions are learned during training, and gradient-free optimization using Evolutionary Algorithms (EAs) to co-optimize network topology and spiking neuron parameters within fixed spatial grids.

6.2.1. Spatial Embedding: Gradient-based Optimization

This section evaluates the gradient-based spatial embedding approach introduced in Section 5.2.1. The analysis first assesses the classification accuracy of spatially embedded MLPs and SNNs on the MNIST benchmark, comparing them against conventional baselines to establish their performance characteristics. Subsequently, it examines the models' robustness to magnitude-based weight pruning, which, due to the inverse relationship between weight and distance (Equation 5.6), corresponds to removing the longest connections.

6.2.1.1. Performance

The performance of the spatial embedding framework was evaluated on the MNIST dataset [50], as described in Section 5.2.1. All models were trained with three different random seeds, and results are reported as mean ± standard deviation.

First, we evaluated a spatially embedded MLP with one hidden layer of 2048 neurons. A model with fixed inter-layer distances of 1.0 achieved a test accuracy of $90.18\% \pm 0.42\%$. Allowing the layer distances to be learnable improved the accuracy to $92.17\% \pm 0.24\%$ with a negligible increase in parameter count from 10,574 to 10,576. As shown in Table 6.19, both configurations were outperformed by the two conventional MLP baselines: a parameter-matched baseline with 14 hidden neurons ($94.29\% \pm 0.28\%$) and a neuron-matched baseline with 2048 hidden neurons ($97.45\% \pm 0.03\%$).

Table 6.19.: Accuracy comparison of the spatially embedded MLP (3D MLP) with 2048 hidden neurons and conventional MLP baselines on the MNIST dataset. Results are reported as mean ± standard deviation over three runs.

Model	Test Accuracy	Parameters
3D MLP (learned dist.)	0.9217 ± 0.0024	10,576
Baseline MLP (14 hid. neurons)	0.9429 ± 0.0028	11,140
Baseline MLP (2048 hid. neurons)	0.9745 ± 0.0003	1,628,170

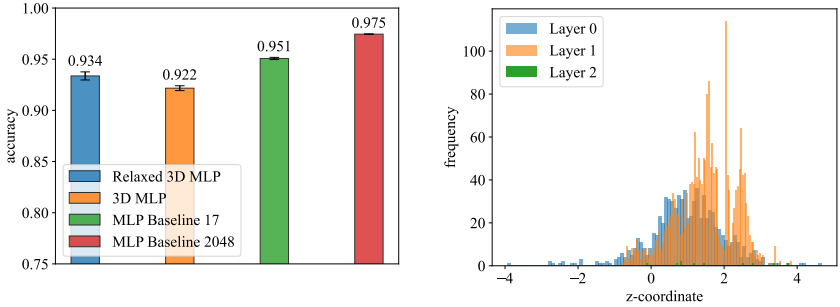
The spatial embedding concept was also applied to SNNs. A 3D SNN with 2048 hidden neurons and learnable layer distances achieved an accuracy of $92.16\% \pm 0.52\%$. As detailed in Table 6.20, this model outperformed the parameter-matched baseline SNN with 14 hidden neurons ($90.31\% \pm 0.88\%$) but was surpassed by the neuron-matched baseline with 2048 hidden neurons ($95.25\% \pm 0.13\%$).

Table 6.20.: Accuracy comparison of the spatially embedded SNN (3D SNN) with 2048 hidden neurons and conventional SNN baselines on the MNIST dataset. Results are reported as mean ± standard deviation over three runs.

Model	Test Accuracy	Parameters
3D SNN (learned dist.)	0.9216 ± 0.0052	10,576
Baseline SNN (14 hid. neurons)	0.9031 ± 0.0088	11,140
Baseline SNN (2048 hid. neurons)	0.9525 ± 0.0013	1,628,170

To investigate the impact of spatial constraints, we evaluated architectural variations. Relaxing the fixed z-coordinate constraint and allowing each neuron to optimize its position along the z-axis improved the MLP's accuracy to $93.37\% \pm 0.40\%$. However, this was still below the performance of a conventional MLP with a comparable parameter count (17 hidden neurons), which achieved $95.07\% \pm 0.11\%$ accuracy. The distribution of the learned z-coordinates and the performance comparison are shown in Figure 6.5. The significant overlap between layer distributions confirms that the model leveraged this additional flexibility.

Furthermore, we explored the effect of embedding dimensionality on model performance. As illustrated in Figure 6.6, increasing the embedding space from 3 to 16 dimensions improved the test accuracy of the spatially embedded MLP, with performance plateauing thereafter. While higher dimensionality provided more degrees of freedom for neuron placement, the spatially embedded models were consistently outperformed by conventional MLPs with similar parameter counts across all tested dimensions.



(a) Performance Comparison **(b)** Z-Coordinate Distribution

Figure 6.5.: (a) Accuracy comparison of the spatially embedded MLP with learned z-coordinates, the standard 3D MLP, and baseline MLPs on the MNIST dataset. (b) Distribution of learned z-coordinates for a spatially embedded MLP where neuron z-positions are optimized during training. The overlap between layer distributions indicates that the model utilized the relaxed spatial hierarchy. Figure adapted from [Erb25].

6.2.1.2. Sparsity Analysis

Pruning experiments were conducted on spatially embedded MLPs with learned layer distances (3D ANN) and compared against two conventional MLP baselines: one with 14 hidden neurons (11,140 parameters) and another with 2048 hidden neurons (1,628,170 parameters). The spatially embedded model contained 10,576 parameters while maintaining the same number of connections (1,626,112) as the larger baseline. All models were evaluated on the MNIST dataset using three random seeds, with performance reported as mean ± standard deviation.

Two pruning strategies were implemented: post-training pruning, where weights are removed after model training is complete, and pruning during training, where the smallest weights are set to zero in each forward pass. Pruning percentages ranged from 20% to 95% of total connections.

Table 6.21 presents the post-training pruning results, visualized in Figure 6.7. Initially, the conventional baseline with 14 neurons outperformed the spatially embedded model. However, at high pruning levels ($\geq$80%),

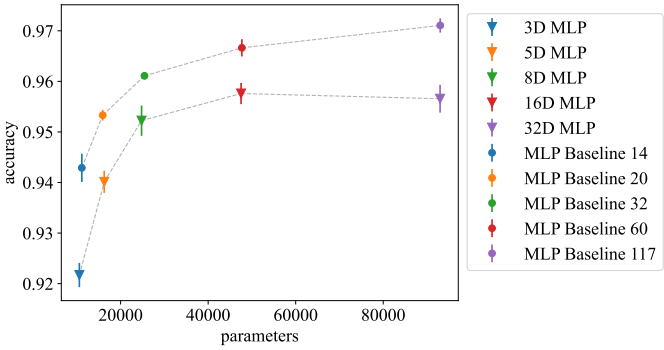


Figure 6.6.: Accuracy versus parameter count for MLPs with 2048 hidden neurons embedded in higher-dimensional spaces (3D to 32D) and their corresponding parameter-matched conventional MLP baselines. Performance is evaluated on the MNIST dataset. Figure adapted from [Erb25].

the 3D ANN demonstrated superior robustness, achieving 60.51% ± 3.17% accuracy at 80% pruning compared to 43.46% ± 8.69% for the 14-neuron baseline. Most remarkably, at 95% pruning, the spatially embedded model (36.24% ± 6.81%) outperformed even the much larger 2048-neuron baseline (27.25% ± 5.53%), despite the baseline having 150× more parameters.

Table 6.22 shows the results when pruning is integrated into the training process, also visually displayed in Figure 6.8. This approach yielded improved performance for the spatially embedded model, achieving 91.70% ± 0.40% accuracy at 20% pruning and maintaining competitive performance even at extreme pruning levels. At 80% pruning, the 3D ANN (92.16% ± 0.61%) significantly outperformed the 14-neuron baseline (86.71% ± 4.10%). At 95% pruning, the spatially embedded model achieved 82.65% ± 3.63% accuracy compared to 41.44% ± 3.08% for the parameter-matched baseline.

These results demonstrate the potential of spatial embedding for achieving high connection sparsity. Post-training pruning at 95% sparsity corresponds to a 20-fold reduction in $N_{\text{connections}}$. Pruning during training is even more effective, achieving 80% sparsity (a 5-fold reduction) with negligible performance loss, and 90% sparsity (a 10-fold reduction) with only

Table 6.21.: Accuracy comparison for magnitude-based weight pruning applied after training. Models were trained using three different random seeds, with accuracy reported as mean ± standard deviation.

Prune Percentage	3D ANN	Baseline 14	Baseline 2048
20%	0.80 ± 0.02	0.94 ± 0.00	**0.97 ± 0.00**
40%	0.73 ± 0.01	0.91 ± 0.01	**0.97 ± 0.00**
60%	0.69 ± 0.08	0.75 ± 0.08	**0.96 ± 0.01**
80%	0.61 ± 0.03	0.43 ± 0.09	**0.83 ± 0.01**
90%	0.51 ± 0.07	0.17 ± 0.03	**0.55 ± 0.08**
95%	**0.36 ± 0.07**	0.09 ± 0.01	0.27 ± 0.06
Parameters	10,576	11,140	1,628,170
Connections	1,626,112	11,116	1,626,112

a minor drop in accuracy. These reductions directly lower the estimated synaptic energy consumption as per Equation 3.10.

6.2.1.3. Discussion

The results reveal a clear trade-off: while spatial embedding reduces parameter complexity from $O(n^2)$ to $O(n)$, it impairs raw classification performance. Spatially embedded MLPs consistently underperformed conventional baselines, even those with comparable parameter counts. This is attributed to inherent weight interdependencies; because weights are tied to neuron positions, a single positional update affects all of a neuron's connections, limiting model expressiveness. Relaxing spatial constraints improved accuracy but did not close the performance gap. Notably, the spatially embedded SNN did outperform its parameter-matched baseline, suggesting temporal dynamics might compensate for reduced weight flexibility.

The primary advantage of this approach is its robustness to pruning. As detailed in the sparsity analysis, spatially embedded models maintain high accuracy at extreme pruning levels. This translates to substantial reductions in the number of non-zero connections ($N_{\text{connections}}$) and thus also in synaptic operations within the energy estimation framework of Equation 3.10. For instance, pruning during training achieved a 5-fold

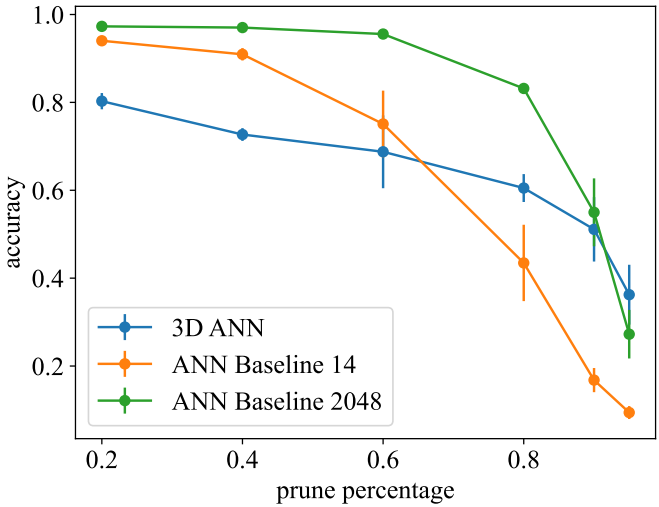


Figure 6.7.: Accuracy comparison for post-training magnitude-based weight pruning on MNIST. The spatially embedded 3D ANN demonstrates superior robustness at high pruning levels ($\geq$90%) compared to conventional MLP baselines. Models were trained using three different random seeds, with error bars representing standard deviation. Figure adapted from [Erb25].

reduction in connections (80% sparsity) with no loss in accuracy, and a 10-fold reduction (90% sparsity) with only a minor performance drop. This resilience arises because pruning removes noisy, long-range connections and reduces weight interdependencies, allowing for more targeted optimization of neuron positions. These findings establish spatial embedding as a potential regularizer for creating sparse, hardware-efficient models.

Table 6.22.: Accuracy comparison for magnitude-based weight pruning applied during training. Models were trained using three different random seeds, with accuracy reported as mean ± standard deviation.

Prune Percentage	3D ANN	Baseline 14	Baseline 2048
20%	0.92 ± 0.00	0.94 ± 0.00	**0.97 ± 0.00**
40%	0.92 ± 0.01	0.94 ± 0.00	**0.97 ± 0.00**
60%	0.92 ± 0.01	0.93 ± 0.00	**0.97 ± 0.00**
80%	0.92 ± 0.01	0.87 ± 0.04	**0.97 ± 0.00**
90%	0.89 ± 0.02	0.64 ± 0.05	**0.96 ± 0.00**
95%	0.83 ± 0.04	0.41 ± 0.03	**0.96 ± 0.00**
Parameters	10,576	11,140	1,628,170
Connections	1,626,112	11,116	1,626,112

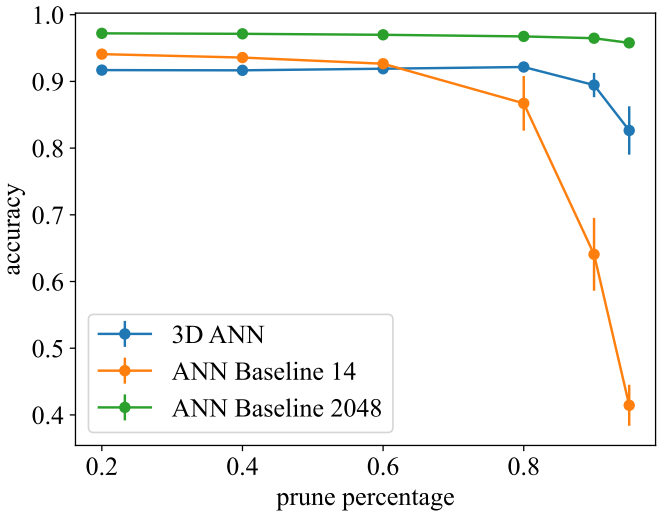


Figure 6.8.: Accuracy comparison for pruning during training on MNIST. The spatially embedded 3D ANN maintains competitive performance across all pruning levels when sparse connectivity is learned from the beginning of training. Models were trained using three different random seeds, with error bars representing standard deviation. Figure adapted from [Erb25].

6.2.2. Spatial Embedding: Gradient-free Optimization

This section evaluates the networks evolved using the gradient-free optimization approach detailed in Section 5.2.2. In contrast to gradient-based methods, this EA-based technique co-optimizes network topology and individual neuron parameters. It treats a network's spatial embedding as a fixed architectural constraint: neurons are positioned in predefined grids, and connectivity is governed by Euclidean distance.

This spatial arrangement inherently promotes sparse, locally connected networks. As described in Section 5.2.2, the probability of a connection between neurons decreases with their distance, and a scaling factor attenuates the strength of long-range connections. The effect is a high number of near-zero weights, as illustrated by the connectivity matrices for different spatial dimensionalities in Figure 5.16. The following subsections analyze the resulting task performance and parameter efficiency in detail.

6.2.2.1. Performance

To evaluate task performance, the evolved networks were tested across five continuous control environments from the Gymnasium library [94]. As detailed in Section 5.2.2, four distinct 64-neuron configurations were compared: a non-embedded baseline (0D) and three spatially embedded topologies (1D, 2D, and 3D).

Figure 6.9 shows that spatially embedded networks consistently outperformed their non-embedded counterparts. Among the embedded configurations, the 2D topology generally yielded the best performance, particularly in less complex environments. The 1D and 3D embeddings also proved effective, demonstrating strong performance in specific contexts.

A comparative analysis with State-of-the-Art (SOTA) baseline models from the Farama-Minari repository [7] shows the performance of the evolved networks. As summarized in Table 6.23, the evolved SNNs achieve competitive results. For instance, in the Swimmer environment, the evolved network achieved 99.48% of the baseline performance.

However, a performance gap emerges in more complex environments. In the Ant environment, the evolved network reached 23.04% of the baseline performance. This discrepancy is attributed to the limited size of the

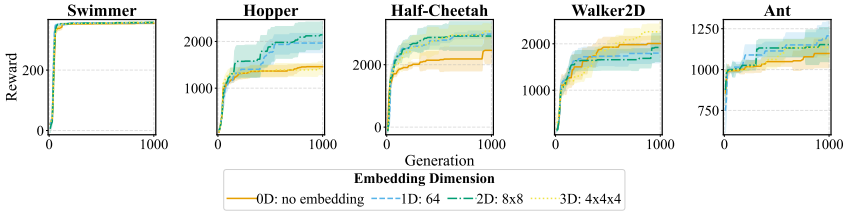


Figure 6.9.: Performance comparison of spatial embeddings across five environments. Plots show test reward over generations, evaluated on 100 unseen seeds. Results are averaged over three independent runs, with shaded regions representing the standard error. Figure adapted from [VSSB25].

evolved networks (64 neurons), suggesting that while the current approach is effective, larger networks may be required to match SOTA performance on more demanding tasks.

Table 6.23.: Performance comparison of spatially embedded SNNs and SOTA baselines reported as mean reward ± standard error. "RP" (Relative Performance) is the percentage of the baseline reward.

Environment	SOTA	Spatial	RP
Swimmer	363.7 ± 1.8	361.8 ± 1.7	99.48%
Hopper	4098.2 ± 247.7	2706.1 ± 73.7	66.03%
HalfCheetah	17641.8 ± 61.9	3670.4 ± 1198.3	20.81%
Walker2d	6956.6 ± 15.9	2492.0 ± 247.7	35.82%
Ant	5846.3 ± 138.5	1346.8 ± 35.1	23.04%

6.2.2.2. Sparsity Analysis

To further enhance sparsity, networks evolved through this method were subjected to post-evolution pruning, where connections with weights below various absolute thresholds were removed. Figure 6.10 demonstrates that this process can yield extremely sparse networks. Across multiple continuous control environments, high levels of sparsity (over 95%) were achieved

with minimal impact on task performance. The 1D embedding, in particular, proved robust to pruning, allowing for the removal of connections with weights below a threshold of 3 without performance degradation in most tested environments.

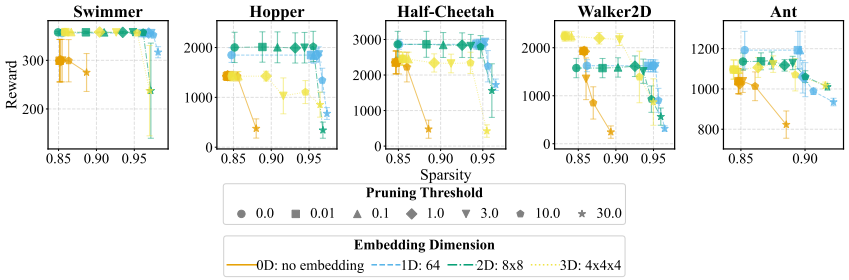


Figure 6.10.: Performance-sparsity trade-off for pruned SNNs across five environments. Markers represent different weight pruning thresholds. Sparsity is defined as the percentage of zero-value weights. The results show that high sparsity can be achieved with little to no drop in performance. Error bars represent standard error over three training runs. Figure adapted from [VSSB25].

The efficiency in terms of non-zero weights resulting from this approach is summarized in Table 6.24. Compared to state-of-the-art deep reinforcement learning models, the evolved, spatially embedded SNNs operate with a drastically reduced number of non-zero weights—as little as 0.06% of the baseline non-zero weights in some cases. This corresponds to a reduction of 98.54% to 99.94% in the number of non-zero weights.

In addition to structural sparsity, the evolved SNNs exhibit high activation sparsity, as shown in Table 6.25. The networks exhibit activation sparsity ranging from 66% to 80%, indicating that, on average, a neuron remains inactive for the majority of timesteps. This temporal sparsity is a key factor in reducing energy consumption, as it directly lowers the number of synaptic operations required per inference step.

Table 6.24.: Comparison of spatially embedded SNNs and SOTA baselines reported as the number of non-zero weights (NZW). "RWN" is the percentage of NZW relative to the baseline. "Dim." is the dimensionality of the spatial embedding. "Pruning" is the pruning threshold.

Environment	SOTA	Spatial	RWN	Dim.	Pruning
Swimmer	9,408	137	1.46 %	2D	30.0
Hopper	347,392	222	0.06 %	2D	10.0
HalfCheetah	384,256	703	0.18 %	2D	0.01
Walker2d	359,680	694	0.19 %	3D	1.0
Ant	475,392	1,208	0.25 %	1D	0.1

Table 6.25.: Spiking activity of the spatially embedded SNNs. Activation sparsity is calculated as $1 -$ (spikes per neuron/timesteps). Values are reported per environment step as mean $\pm$ standard deviation across 100 seeds.

Environment	Spikes	Activation Sparsity
Swimmer	$(9.40 \pm 0.01) \times 10^2$	$80 \pm 0\%$
Hopper	$(1.12 \pm 0.02) \times 10^3$	$77 \pm 0\%$
HalfCheetah	$(1.47 \pm 0.04) \times 10^3$	$72 \pm 1\%$
Walker2d	$(1.69 \pm 0.06) \times 10^3$	$67 \pm 1\%$
Ant	$(3.68 \pm 0.04) \times 10^3$	$66 \pm 0\%$

6.2.2.3. Energy Evaluation

To assess the energy efficiency of the evolved networks, this section analyzes the estimated energy consumption per environment step. The evaluation compares the spatially embedded SNNs on a simulated neuromorphic platform (Intel's Loihi) against the SOTA baseline models on conventional hardware (CPU and ARM). The analysis is based on the synaptic operations resulting from the structural and activation sparsity discussed in the previous section.

The resulting synaptic operations are detailed in Table 6.26. For the SOTA ANNs, which are dense, the number of synaptic operations is equivalent

to the number of non-zero weights and remains constant. In contrast, the synaptic operations for the SNNs are event-driven and significantly lower due to their sparse activity, showing reductions of up to almost 50× compared to the baselines. Despite the high connection and activation sparsity, the SNNs still require a significant number of synaptic operations. This is due to the fact that the SNN performs 64 forward passes per environment step to generate spiking activity, compared to the 1 forward pass for the SOTA ANNs. Nevertheless, this is a hyperparameter that can be tuned to further reduce the number of synaptic operations.

Table 6.26.: Synaptic operations comparison of SOTA and spatially embedded SNN models per environment step. Values for SNNs are reported as mean ± standard deviation.

Environment	**Synaptic Operations**	
	SOTA	**Spatial**
Swimmer	9.41×10^3	$\mathbf{(5.49 \pm 0.00) \times 10^3}$
Hopper	3.47×10^5	$\mathbf{(7.96 \pm 0.18) \times 10^3}$
HalfCheetah	3.84×10^5	$\mathbf{(1.54 \pm 0.04) \times 10^4}$
Walker2d	3.60×10^5	$\mathbf{(1.72 \pm 0.05) \times 10^4}$
Ant	4.75×10^5	$\mathbf{(3.84 \pm 0.03) \times 10^4}$

Based on these activity and synaptic operation counts, the estimated energy consumption is presented in Table 6.27. The results show a significant reduction in estimated energy consumption for the SNNs on Loihi. Compared to the SOTA models on an ARM processor, the SNN on Loihi reaches as much as 500× less energy consumption. This advantage grows to upwards of 5,000× when comparing against a conventional CPU. This highlights the combined benefit of algorithmic innovations and specialized hardware.

6.2.2.4. Discussion

The findings from the gradient-free optimization approach highlight a trade-off between task performance and network sparsity. The evolved, spatially embedded networks demonstrate competitive performance against SOTA

Table 6.27.: Energy comparison of SOTA and spatially embedded SNN models per environment step. Values for SNNs are reported as mean ± standard deviation.

Environment	Model	Energy (J)	Platform
Swimmer	SOTA	8.31×10^{-5}	CPU
		8.70×10^{-6}	ARM
	Spatial	$\mathbf{(4.91 \pm 0.00) \times 10^{-7}}$	Loihi
Hopper	SOTA	3.01×10^{-3}	CPU
		3.15×10^{-4}	ARM
	Spatial	$\mathbf{(5.63 \pm 0.00) \times 10^{-7}}$	Loihi
HalfCheetah	SOTA	3.33×10^{-3}	CPU
		3.48×10^{-4}	ARM
	Spatial	$\mathbf{(7.79 \pm 0.01) \times 10^{-7}}$	Loihi
Walker2d	SOTA	3.12×10^{-3}	CPU
		3.26×10^{-4}	ARM
	Spatial	$\mathbf{(8.28 \pm 0.01) \times 10^{-7}}$	Loihi
Ant	SOTA	4.11×10^{-3}	CPU
		4.30×10^{-4}	ARM
	Spatial	$\mathbf{(1.41 \pm 0.01) \times 10^{-6}}$	Loihi

baselines, particularly on tasks of lower complexity, as shown in Table 6.23. For instance, the network for the Swimmer environment achieves 99.48% of the SOTA performance. While a performance gap is evident on more complex tasks, this is largely attributable to the constrained network size of 64 neurons.

A key advantage of this method is its ability to produce these results with extreme efficiency in terms of non-zero weights. Post-evolution pruning, as illustrated in Figure 6.10, achieves over 95% sparsity with minimal impact on performance. This efficiency is further quantified in Table 6.24, which shows that the evolved networks operate with as little as 0.06% of the non-zero weights used by SOTA models.

In the context of Equation 3.10, this reduction in the number of non-zero weights ($N_{\text{connections}}$)—ranging from a 70-fold to over a 1600-fold

decrease—translates into a substantial reduction in the estimated energy consumption, especially when paired with a neuromorphic platform, resulting in energy savings of up to 3.5 orders of magnitude. This result indicates the potential of gradient-free spatial embedding for creating efficient controllers suitable for resource-constrained applications.

7. Discussion

This dissertation was motivated by the need for energy-efficient neural networks. It was guided by the principle that the energy consumption of synaptic operations can be approximated as $E \approx E_{\text{synop}} \times N_{\text{spikes}} \times N_{\text{connections}}$. This framework decomposes the problem into improving hardware efficiency (E_{synop}) and reducing software-level complexity through activation sparsity (N_{spikes}) and connection sparsity ($N_{\text{connections}}$). This chapter synthesizes the results of the software-level strategies explored in this work, discusses their limitations, and outlines directions for future research.

7.1. Software-Level Sparsity for Energy Efficiency

The research presented in this dissertation demonstrates that targeting activation and connection sparsity through software-level design is a viable strategy for reducing the energy footprint of neural networks.

Activation sparsity (N_{spikes}) was addressed through two main strategies. First, a spike encoding framework was developed to convert continuous signals into sparse spike trains at the network's input, reducing data density by up to 5.6× (Section 6.1.1). Second, integrating spiking neurons into conventional CNN and RNN architectures consistently yielded high activation sparsity. In the sleep analysis task (Section 6.1.2), this produced an average activation sparsity of 84.25% (a 6× reduction). For the neural decoding and predictive maintenance tasks (Sections 6.1.3 and 6.1.4), introducing LIF units into recurrent architectures resulted in activation sparsities of up to 94% (16× reduction) and 90.25% (10× reduction), respectively. Collectively, these techniques produced activation sparsities in the range of 84–94%, reducing the number of spike-driven computations by a factor of 6× to 16×.

Connection sparsity ($N_{\text{connections}}$) was addressed by methods that treat sparse connectivity as an intrinsic design principle. When the gradient-based spatial embedding method (Section 6.2.1) was combined with pruning during training, it achieved an 80% reduction in connections (a 5-fold decrease) with almost no loss in accuracy. The gradient-free evolutionary approach (Section 6.2.2) produced parameter-efficient controllers that operated with up to 99.94% fewer non-zero weights than state-of-the-art baselines—a reduction of over 1600×—while maintaining competitive performance on control tasks of higher complexity and matching the performance of lower complexity tasks.

These software-level strategies for inducing sparsity can, on their own, lead to estimated energy savings of one to three orders of magnitude. When accounting for both the hardware efficiency of neuromorphic platforms such as Loihi and the additional simulation timesteps required by SNNs—which can offset some of the gains—the total estimated energy savings can reach up to 5000× relative to a conventional x86 CPU and 500× relative to a ARM CPU.

7.2. Limitations and Future Directions

Despite these results, this work has several limitations that define clear directions for future research. A primary limitation is that the energy savings are theoretical, based on an approximated energy model. Furthermore, performance gaps remain when the proposed models are compared to state-of-the-art dense ANNs on the most complex tasks, especially in the evolved spatially embedded SNN controllers (Section 6.2.2).

A key challenge identified is the trade-off between the computational cost of temporal simulation and the efficiency gains from sparsity. For instance, the sleep analysis CNN (Section 6.1.2) required $T = 10$ timesteps to achieve its accuracy, while the evolved controllers (Section 6.2.2) used $T = 64$. This overhead, which is necessary for simple neuron models like the LIF to integrate information, can offset some of the benefits of sparse computation.

This trade-off points toward several future research directions. The first is the adoption of more expressive neuron models that go beyond simple

rate-based signaling to use temporal coding, where information is conveyed in the precise timing of a few spikes. This aligns with the idea that a system's expressive power is defined not by its state of activity at any given moment, but by the vast repertoire of states it *could* potentially enter [36]. In this view, the temporal dimension is not an overhead but a resource for encoding information. For a fixed energy budget—that is, a constant number of synaptic operations per inference—one can expand the network's expressive capacity by increasing the number of simulation timesteps or the number of potential (but inactive) connections. A single spike, timed precisely within one of a million timesteps, can carry more information than a single spike within one of ten. This suggests a path toward scaling a system's *potential* complexity (expressivity) super-linearly, while its *realized* complexity (the energy consumed) scales linearly, due to the increased maintenance cost of the larger number of neurons.

A second, related challenge is the development of training methods that can leverage this large potential state space efficiently, similarly to how EventProp [105] scales its memory requirements for training SNNs linearly with the number of events $\mathcal{O}(N_{\text{events}})$. Future work should focus on the co-design of learning rules, software frameworks and hardware platforms that allow for training algorithms whose computational complexity scales with the number of *actual* spikes and active connections, not with the number of *potential* ones.

A third direction is to incorporate the energy estimation model from Chapter 3 directly into the loss function. This would enable the training process to directly optimize for energy efficiency alongside task performance, rather than treating sparsity as a proxy for low energy.

Finally, a necessary next step is to validate the theoretical energy estimations presented in this dissertation through direct power measurements on physical neuromorphic hardware. This would provide empirical confirmation of the real-world efficiency gains achievable through the proposed software-level sparsity strategies.

7.3. Information, Scaling, and Sparsity

The trade-off between a network's expressive capacity and its energy consumption can be analyzed from an information-theoretic perspective. The expressive capacity relates to the number of potential states a network can assume, while the energy cost is tied to the subset of states that are actually realized during computation. This section formalizes this relationship by quantifying the information capacity of a neural code, defined as the logarithm of the number of distinct possible network states [88]. This framework is used to analyze how capacity and its associated energy cost scale with network size under different sparsity assumptions.

An Information-Theoretic Model To construct the model, we first define the state space of the network. Let us consider a network with N_{neurons} per layer and a simulation window of T discrete timesteps. The potential state space can be decomposed into three components:

- **Structural State Space:** The number of ways to choose which k connections are active from the $N_C = N_{\text{neurons}}^2$ possible locations:

$$M_{\text{struct}} = \binom{N_C}{k} = \binom{N_{\text{neurons}}^2}{k} \tag{7.1}$$

- **Weight State Space:** For each of the k active connections, its weight can be represented by 32 bits. The number of possible weight configurations is:

$$M_{\text{weights}} = (2^{32})^k = 2^{32k} \tag{7.2}$$

- **Activity State Space:** The number of ways to choose which s spike slots are active from the $N_S = N_{\text{neurons}} \times T$ possible slots:

$$M_{\text{act}} = \binom{N_S}{s} = \binom{N_{\text{neurons}} T}{s} \tag{7.3}$$

Assuming these choices are independent, the total information capacity I is the sum of the logarithms of the number of states in each space:

$$
\begin{aligned}
I &= \log_2(M_{\text{struct}}) + \log_2(M_{\text{weights}}) + \log_2(M_{\text{act}}) \\
&= \log_2\left(\binom{N_{\text{neurons}}^2}{k}\right) + 32k + \log_2\left(\binom{N_{\text{neurons}}T}{s}\right)
\end{aligned} \tag{7.4}
$$

Approximation for the Sparse Regime In a biologically plausible sparse regime, where $k \ll N_{\text{neurons}}^2$ and $s \ll N_{\text{neurons}}T$, the combinatorial terms in Equation 7.4 can be approximated. This is done in two steps.

First, the logarithm of the binomial coefficient can be related to the binary entropy function using Stirling's approximation for factorials ($\ln n! \approx n \ln n - n$):

$$
\begin{aligned}
\ln \binom{N}{k} &= \ln(N!) - \ln(k!) - \ln((N-k)!) \\
&\approx (N \ln N - N) - (k \ln k - k) - ((N-k)\ln(N-k) - (N-k)) \\
&= N \ln N - k \ln k - (N-k)\ln(N-k)
\end{aligned}
$$

By substituting $k = pN$, where $p = k/N$ is the density, we can rearrange the terms to show that $\ln \binom{N}{k} \approx N H_e(p)$, where $H_e(p)$ is the binary entropy function with the natural logarithm. Converting to base-2 gives the widely used approximation:

$$
\log_2 \binom{N}{k} \approx N H_2(k/N) \tag{7.5}
$$

where $H_2(p) = -p \log_2 p - (1-p)\log_2(1-p)$.

Second, for the sparse regime where the density p is very small ($p \to 0$), the binary entropy function $H_2(p)$ simplifies to $H_2(p) \approx -p \log_2 p$. The approximation for the full expression becomes:

$$
N H_2(k/N) \approx N\left(-\frac{k}{N}\log_2\frac{k}{N}\right) = k \log_2 \frac{N}{k} \tag{7.6}
$$

Applying this approximation to the combinatorial terms in Equation 7.4 yields the final expression for information capacity in the sparse regime:

$$I \approx k \log_2 \left(\frac{N_{\text{neurons}}^2}{k} \right) + 32k + s \log_2 \left(\frac{N_{\text{neurons}} \times T}{s} \right) \tag{7.7}$$

Analysis of Scaling Strategies This formulation provides a basis for analyzing two distinct network scaling strategies.

Constant-Density Scaling. A conventional approach is to scale a network while keeping the connection density $\alpha = k/N_{\text{neurons}}^2$ and the average number of spikes per neuron constant. This implies that the total number of connections scales as $k \propto N_{\text{neurons}}^2$ and the total number of spikes scales as $s \propto N_{\text{neurons}}$. The total information capacity, $I = I_{\text{struct}} + I_{\text{weights}} + I_{\text{activity}}$, therefore scales as:

$$I \propto N_{\text{neurons}}^2 H_2(\alpha) + 32(\alpha N_{\text{neurons}}^2) + N_{\text{neurons}} T H_2(\beta) \approx O(N_{\text{neurons}}^2) \tag{7.8}$$

The energy cost, using the model from Chapter 3 where energy is proportional to the product of connections and the average number of spikes per neuron ($E \propto k \cdot (s/N_{\text{neurons}})$), scales as:

$$E \propto (N_{\text{neurons}}^2) \cdot (N_{\text{neurons}}/N_{\text{neurons}}) \propto N_{\text{neurons}}^2 \tag{7.9}$$

Consequently, the information-per-energy ratio $\eta = I/E$ approaches a constant, indicating no efficiency improvement with scale.

Sub-linear Scaling. In contrast, a strategy analogous to that observed in the brain is to increase the potential state space (larger N_{neurons}) while allowing the number of actualized elements, k and s, to grow sub-linearly. We assume that $k \propto N_{\text{neurons}}^{\gamma}$ and $s \propto N_{\text{neurons}}^{\delta}$, for exponents such as $0 < \gamma < 2$ and $0 \leq \delta < 1$.

The information contribution from connections, I_k, can be derived by substituting the scaling of k into the sparse approximation:

$$I_k = k \log_2 \left(\frac{N_{\text{neurons}}^2}{k} \right) \propto N_{\text{neurons}}^{\gamma} \log_2 \left(\frac{N_{\text{neurons}}^2}{N_{\text{neurons}}^{\gamma}} \right)$$
$$= (2-\gamma) N_{\text{neurons}}^{\gamma} \log_2(N_{\text{neurons}})$$
$$\propto N_{\text{neurons}}^{\gamma} \log(N_{\text{neurons}})$$

Similarly, for the spike information term, I_s, assuming T is constant or grows slower than N_{neurons}:

$$I_s = s \log_2 \left(\frac{N_{\text{neurons}} T}{s} \right) \propto N_{\text{neurons}}^{\delta} \log_2 \left(\frac{N_{\text{neurons}} T}{N_{\text{neurons}}^{\delta}} \right)$$
$$\propto (1-\delta) N_{\text{neurons}}^{\delta} \log_2(N_{\text{neurons}})$$
$$\propto N_{\text{neurons}}^{\delta} \log(N_{\text{neurons}})$$

The total information is the sum of three contributions, $I = I_k + I_s + I_{\text{weights}}$, where $I_{\text{weights}} = 32k \propto N_{\text{neurons}}^{\gamma}$. The energy cost is given by:

$$E \propto k \cdot (s/N_{\text{neurons}}) \propto N_{\text{neurons}}^{\gamma+\delta-1} \tag{7.10}$$

We can analyze the scaling of the information-per-energy ratio, η, by considering three cases for the dominant scaling exponent. Because the total information I is a sum of terms with different scaling behaviors, we must first determine the dominant term for I based on the relationship between γ and δ. This requires a case-by-case analysis.

Case 1: Connection scaling dominates ($\gamma > \delta$). In this case, the total information I is a sum of terms proportional to $N_{\text{neurons}}^{\gamma} \log(N_{\text{neurons}})$, $N_{\text{neurons}}^{\delta} \log(N_{\text{neurons}})$, and $N_{\text{neurons}}^{\gamma}$. For large N_{neurons}, the term $N_{\text{neurons}}^{\gamma} \log(N_{\text{neurons}})$ grows asymptotically faster than both other terms. The overall information scaling is therefore dominated by the structural information:

$$I \propto N_{\text{neurons}}^{\gamma} \log(N_{\text{neurons}}) \tag{7.11}$$

The information-per-energy ratio is then:

$$\eta = \frac{I}{E} \propto \frac{N_{\text{neurons}}^{\gamma} \log N_{\text{neurons}}}{N_{\text{neurons}}^{\gamma+\delta-1}} = N_{\text{neurons}}^{1-\delta} \log N_{\text{neurons}} \tag{7.12}$$

Given the assumption that $0 \leq \delta < 1$, the exponent $1 - \delta$ is positive, indicating a polynomial improvement in efficiency.

Case 2: Spike scaling dominates ($\delta > \gamma$). Similarly, for large N_{neurons}, the $N_{\text{neurons}}^{\delta} \log(N_{\text{neurons}})$ term grows asymptotically faster than the other two. The information scaling is now dominated by the activity information:

$$I \propto N_{\text{neurons}}^{\delta} \log(N_{\text{neurons}}) \tag{7.13}$$

The information-per-energy ratio is:

$$\eta = \frac{I}{E} \propto \frac{N_{\text{neurons}}^{\delta} \log N_{\text{neurons}}}{N_{\text{neurons}}^{\gamma+\delta-1}} = N_{\text{neurons}}^{1-\gamma} \log N_{\text{neurons}} \tag{7.14}$$

Given that $\delta > \gamma$ and our assumption that $\delta < 1$, it follows that γ must also be less than 1. Therefore, the exponent $1 - \gamma$ is also positive, again indicating a polynomial improvement in efficiency.

Case 3: Equal scaling exponents ($\gamma = \delta$). In this scenario, the structural and activity information terms (I_k, I_s) scale as $N_{\text{neurons}}^{\gamma} \log(N_{\text{neurons}})$, while the weight information (I_{weights}) scales as $N_{\text{neurons}}^{\gamma}$. For large N_{neurons}, the logarithmic factor makes the structural and activity terms dominant. The total information scaling is therefore:

$$I \propto N_{\text{neurons}}^{\gamma} \log(N_{\text{neurons}}) \tag{7.15}$$

The energy scaling becomes:

$$E \propto N_{\text{neurons}}^{2\gamma-1}$$

The information-per-energy ratio is therefore:

$$\eta = \frac{I}{E} \propto \frac{N_{\text{neurons}}^{\gamma} \log N_{\text{neurons}}}{N_{\text{neurons}}^{2\gamma-1}} = N_{\text{neurons}}^{1-\gamma} \log N_{\text{neurons}} \tag{7.16}$$

Since the condition $\delta < 1$ implies $\gamma < 1$ in this case, the exponent $1 - \gamma$ is positive, confirming a polynomial improvement in efficiency.

In all three cases, the sub-linear strategy yields a polynomial gain in information-per-energy as the network scales. This provides a quantitative basis for the energy advantages of sparse, brain-like architectures.

A Quantitative Example A concrete example illustrates this trade-off, comparing two networks with the same information capacity.

Case A: Baseline Network. Consider a network with $N_{\text{neurons}} = 1,000$, $T = 50$, a connection density of 10% ($k = 100,000$), and a total of $s = 2,500$ spikes. Its total information capacity is the sum of its structural, weight, and activity information:

$$\begin{aligned} I_{\text{struct}} &\approx (1000^2) \cdot H_2(0.1) \approx 469 \text{ kilobits} \\ I_{\text{weights}} &= 32 \cdot 100,000 = 3,200 \text{ kilobits} \\ I_{\text{activity}} &\approx (1000 \cdot 50) \cdot H_2(2500/50000) \approx 14 \text{ kilobits} \\ I_{\text{total}} &\approx 469 + 3200 + 14 = 3,683 \text{ kilobits} \end{aligned}$$

Based on the energy model from Chapter 3 ($E \propto k \cdot (s/N_{\text{neurons}})$), its associated energy cost is:

$$E_A \propto 100,000 \cdot (2,500/1,000) = 250,000 \text{ units}$$

Case B: Scaled Sparse Network. Now consider a larger network with $N_{\text{neurons}} = 10,000$ that maintains the same total information capacity ($I_{\text{total}} \approx 3,683$ kilobits) and the same total spike count ($s = 2,500$). The activity information for this larger network increases to $I_{\text{activity}} \approx 23$ kilobits. To match the total capacity, the sum of structural and weight information must be $I_{\text{struct}} + I_{\text{weights}} \approx 3,660$ kilobits. This is achieved with approximately $k \approx 83,830$ connections, which provide

$I_{\text{weights}} \approx 2,683$ kilobits and $I_{\text{struct}} \approx 977$ kilobits. The energy cost for this larger, sparser network is:

$$E_B \propto 83,830 \cdot (2,500/10,000) \approx 20,958 \text{ units}$$

Comparison. To achieve the same expressive power, the scaled sparse network requires an energy cost of only 20,958 units, compared to 250,000 units for the baseline network. This represents an energy reduction of approximately 92%. This illustrates a path toward scaling network capacity that contrasts with the dense architectures of current large language models, suggesting that efficiency can be improved by embracing sparsity in high-dimensional state spaces.

8. Conclusion

This dissertation was driven by the need for energy-efficient neural networks capable of deployment on resource-constrained edge devices. The central thesis of this work is that part of the path to such efficiency lies in targeting the software-level complexity of SNNs. This was formalized through an energy estimation framework, $E \approx E_{\text{synop}} \times N_{\text{spikes}} \times N_{\text{connections}}$ (Chapter 3), which guided the research toward two fundamental goals: minimizing neuron activations and reducing synaptic connections.

To address activation sparsity, this research explored methods at both the input and architectural levels (Section 5.1 and 6.1). A flexible software framework for spike encoding was developed, enabling the conversion of continuous data into sparse, event-based representations, while minimizing information loss. Within the network, conventional activation functions in deep learning models were replaced with spiking neuron dynamics, demonstrating that the principles of event-driven computation can be integrated into established architectures like CNNs and RNNs. These hybrid models consistently achieve high activation sparsity across various applications, including sleep analysis and neural decoding. Notably, they have achieved new state-of-the-art performance in two of the three addressed tasks.

To achieve connection sparsity, this work introduced methods inspired by the spatial organization of biological brains (Section 5.2 and 6.2). By embedding neurons in Euclidean space and making their connectivity dependent on distance, this research moved beyond traditional pruning techniques. Both gradient-based and gradient-free optimization methods were developed to create networks where sparse, local connectivity is an intrinsic property. These spatially constrained models, particularly when optimized with evolutionary algorithms, produced highly parameter-efficient solutions for complex control tasks.

The validation of these methods across diverse, real-world problems — spanning predictive maintenance, biomedical signal processing, and robotics —

confirms that a focus on software-level sparsity provides a robust strategy for developing energy-efficient neuromorphic systems. The principles and techniques presented here offer a pathway toward building artificial intelligence that is not only computationally powerful but also sustainable and practical for deployment at the edge.

Looking forward, more work is required to advance the development of brain-inspired artificial intelligence. While this work has demonstrated the potential of sparsity, future research must bridge the gap between theoretical energy estimations and real-world power consumption on neuromorphic hardware. A key challenge lies in overcoming the computational cost of temporal simulation, which can be addressed by developing more sophisticated neuron models that leverage temporal coding for richer information representation. Ultimately, the co-design of learning algorithms, network architectures, and hardware platforms will be essential to create systems that can learn and operate with the efficiency of the brain.

In conclusion, this dissertation has laid a foundation for designing and optimizing SNNs through software-level sparsity. It provides both a conceptual framework and a set of practical methods for building the next generation of intelligent systems that are not only defined by their scale, but also by their efficiency.

A. Appendix

Algorithm A.1 Step Forward (SF) Encoding Algorithm

Input: $signal$, $threshold$
Output: $spike\_train$
$base \leftarrow 0$, $up\_spikes \leftarrow 0$, $down\_spikes \leftarrow 0$
for $t = 1$ to $time\_steps$ **do**
    if $signal(t) > base + threshold$ **then**
        $up\_spikes(t) \leftarrow 1$
        $base \leftarrow base + threshold$
    else if $signal(t) < base - threshold$ **then**
        $down\_spikes(t) \leftarrow -1$
        $base \leftarrow base - threshold$
    else
        $up\_spikes(t) \leftarrow 0$
        $down\_spikes(t) \leftarrow 0$
    end if
end for
$spikes \leftarrow up\_spikes + down\_spikes$

Table A.1.: Energy per Operation (J) for different hardware platforms.

Device	Synop Energy	Neurop Energy	Source
CPU x86 (i7-4960X)	8.60×10^{-9}	8.60×10^{-9}	[22]
CPU ARM (Cortex-A5)	9.00×10^{-10}	9.00×10^{-10}	[22]
GPU (GTX 700 Titan)	3.00×10^{-10}	3.00×10^{-10}	[22]
SpiNNaker	1.33×10^{-8}	2.60×10^{-8}	[37]
SpiNNaker 2	4.50×10^{-10}	2.19×10^{-9}	[37]
Loihi	2.71×10^{-11}	8.10×10^{-11}	[19]

Algorithm A.2 Leaky Integrate-and-Fire (LIF) Encoding Algorithm

Input: *signal*, *threshold*, *membrane_constant*
Output: *spike_train*
signal ← min_max_normalize(*signal*)
signal ← *signal* × 2 − 1
voltage ← 0, *up_spikes* ← 0, *down_spikes* ← 0
for $t = 1$ to *time_steps* **do**
    voltage ← *voltage* + *signal*(t)
    if *voltage* > *threshold* **then**
        up_spikes(t) ← 1
        voltage ← 0
    else if *voltage* < −*threshold* **then**
        down_spikes(t) ← −1
        voltage ← 0
    else
        up_spikes(t) ← 0
        down_spikes(t) ← 0
    end if
    voltage ← *voltage* × *membrane_constant*
end for
spikes ← *up_spikes* + *down_spikes*

Algorithm A.3 Ben's Spiker Algorithm (BSA) Encoding

Input: $signal$, $filter\_order$, $filter\_cutoff$, $threshold$
Output: $spike\_train$
$signal \leftarrow \text{normalize}(signal)$
$fir\_coeff \leftarrow \text{fir\_filter}(filter\_size = filter\_order + 1, filter\_cutoff, sampling\_frequency = 1)$
$spikes \leftarrow 0$
for $t = 1$ to $time\_steps$ **do**
    $err1 \leftarrow 0$
    $err2 \leftarrow 0$
    for $j = 1$ to $filter\_size$ **do**
        if $t + j - 1 \leq time\_steps$ **then**
            $err1 \leftarrow err1 + |signal(t + j - 1) - fir\_coeff(j)|$
            $err2 \leftarrow err2 + |signal(t + j - 1)|$
        end if
    end for
    if $error1 \leq err2 - threshold$ **then**
        $spikes(t) \leftarrow 1$
        for $j = 1$ to $filter\_size$ **do**
            if $t + j - 1 \leq time\_steps$ **then**
                $signal(t + j - 1) \leftarrow signal(t + j - 1) - fir\_coeff(j)$
            end if
        end for
    else
        $spikes(t) \leftarrow 0$
    end if
end for

Algorithm A.4 Pulse Width Modulation (PWM) Encoding Algorithm

Input: $signal$, $frequency$, $downspike$ (boolean)
Output: $spike\_train$
$signal \leftarrow \text{normalize}(signal)$
$carrier \leftarrow \text{sawtooth}(frequency)$
$neg\_carrier \leftarrow 1 - carrier$
$pwm \leftarrow 0$, $up\_spikes \leftarrow 0$, $down\_spikes \leftarrow 0$
for $t = 1$ to $time\_steps$ **do**
    if $signal(t) < carrier(t)$ **then**
        $pwm(t) \leftarrow 1$
    else if $signal(t) > neg\_carrier(t)$ **and** $downspike = \text{True}$ **then**
        $pwm(t) \leftarrow -1$
    else
        $pwm(t) \leftarrow 0$
    end if
end for
for $t = 2$ to $time\_steps$ **do**
    if $pwm(t) = 1$ **and** $pwm(t-1) \neq 1$ **then**
        $up\_spikes(t) \leftarrow 1$
    else if $pwm(t) = -1$ **and** $pwm(t-1) \neq -1$ **then**
        $down\_spikes(t) \leftarrow -1$
    end if
end for
$spikes \leftarrow up\_spikes + down\_spikes$

Algorithm A.5 BSizeNewData

function BSizeNewData($\mathbf{B}_{\text{new\_data\_update}}, \mathbf{K}_{\text{conv}}$)
  $\mathbf{B}_{\text{new\_data}} \leftarrow []$
  for $i = 0$ to length($\mathbf{B}_{\text{new\_data\_update}}$) - 1 **do**
    if i is even **then**
      $B \leftarrow B_{\text{new\_data\_update},i} - 1 + K_{\text{conv},i/2}$
      Append B to $\mathbf{B}_{\text{new\_data}}$
    else
      Append $B_{\text{new\_data\_update},i}$ to $\mathbf{B}_{\text{new\_data}}$
    end if
  end for
  return $\mathbf{B}_{\text{new\_data}}$
end function

Algorithm A.6 Buffer Sizes Calculation

Input: Number of layers N_{conv}, N_{pool}
Input: Convolution kernel sizes $\mathbf{K}_{\text{conv}}$, strides $\mathbf{S}_{\text{conv}}$
Input: Pooling kernel sizes $\mathbf{K}_{\text{pool}}$, strides $\mathbf{S}_{\text{pool}}$
Output: buffer size keypoints, buffer size new data, buffer size new data update

Initialize $R \leftarrow 1$, $B_{\text{update}} \leftarrow 1$, $\mathbf{R}_{\text{list}} \leftarrow []$, $\mathbf{B}_{\text{update\_list}} \leftarrow []$
for $i = 0$ to $N_{\text{conv}} + N_{\text{pool}} - 1$ **do**
  if i is even **then** ▷ Convolutional layer
    $R \leftarrow (K_{\text{conv},i/2} - 1) \cdot B_{\text{update}} + R$
    $B_{\text{update}} \leftarrow B_{\text{update}} \cdot S_{\text{conv},i/2}$
  else ▷ Pooling layer
    $R \leftarrow (K_{\text{pool},(i-1)/2} - 1) \cdot B_{\text{update}} + R$
    $B_{\text{update}} \leftarrow B_{\text{update}} \cdot S_{\text{pool},(i-1)/2}$
  end if
  Append R to $\mathbf{R}_{\text{list}}$, B_{update} to $\mathbf{B}_{\text{update\_list}}$
end for
$\mathbf{B}_{\text{keypoints}} \leftarrow$ BSizeKeypoints($R, \mathbf{K}_{\text{conv}}, \mathbf{S}_{\text{pool}}$)
$\mathbf{B}_{\text{new\_data\_update}} \leftarrow$ BSizeNewDataUpdate($\mathbf{B}_{\text{update\_list}}$)
$\mathbf{B}_{\text{new\_data}} \leftarrow$ BSizeNewData($\mathbf{B}_{\text{new\_data\_update}}, \mathbf{K}_{\text{conv}}$)
return $\mathbf{B}_{\text{keypoints}}, \mathbf{B}_{\text{new\_data}}, \mathbf{B}_{\text{new\_data\_update}}$

Algorithm A.7 BSizeNewDataUpdate

function BSizeNewDataUpdate($\mathbf{B}_{\text{update\_list}}$)
    $\mathbf{B}_{\text{update}} \leftarrow \mathbf{B}_{\text{update\_list}}$
    Append last element of $\mathbf{B}_{\text{update}}$ to itself
    Reverse $\mathbf{B}_{\text{update}}$ and remove its last element
    return $\mathbf{B}_{\text{update}}$
end function

Algorithm A.8 BSizeKeypoints

function BSizeKeypoints($R, \mathbf{K}_{\text{conv}}, \mathbf{S}_{\text{pool}}$)
    $C \leftarrow R$, $\mathbf{B}_{\text{keypoints}} \leftarrow []$
    for $i = 0$ to $2 \cdot \text{length}(\mathbf{K}_{\text{conv}}) - 1$ **do**
        if i is even **then**
            Append C to $\mathbf{B}_{\text{keypoints}}$
        else
            $C \leftarrow C - K_{\text{conv},i/2} + 1$
            Append C to $\mathbf{B}_{\text{keypoints}}$
            $C \leftarrow C/S_{\text{pool},i/2}$
        end if
    end for
    return $\mathbf{B}_{\text{keypoints}}$
end function

Table A.2.: Performance Comparison with State-of-the-Art Methods for ECG-based Sleep Stage and Apnea Classification

Author	Signal	Class	Method	Accuracy	$F1^M$
[104]	ECG^s	2-class	ANN	0.8948	0.7595
[78]	ECG/HR	2-class	ANN	0.8797	0.7940
Our work	ECG^s	2 class	SNN	**0.9469**	0.9268
Our work	ECG^s	2 class	ANN	0.9169	**0.9451**
[78]	ECG/HR	3-class	ANN	0.8013	0.7386
[100]	ECG/HRV	3-class	SVM	0.7351	-
[104]	ECG^s	3-class	ANN	0.8407	0.7174
[111]	ECG^s	3-class	RF	**0.8711**	-
[96]	ECG^s	3-class	ANN	0.8640	0.7867
Our work	ECG^s	3-class	SNN	0.8496	**0.8161**
Our work	ECG^s	3-class	ANN	0.8213	0.7850
[96]	ECG^s	5-class	ANN	**0.7420**	0.6160
[104]	ECG^s	5-class	ANN	0.7116	0.5766
Our work	ECG^s	5-class	SNN	0.6877	**0.6843**
Our work	ECG^s	5-class	ANN	0.6456	0.6272
[17]	ECG^s	SA	ANN	0.8790	-
[60]	ECG^s	SA	ANN	0.8186	0.6963
[67]	ECG	SA	ANN	0.8230	-
[66]	ECG	SA	ANN	0.8558	0.8467
[115]	ECG^s	SA	ANN	0.9610	-
[27]	ECG	SA	ANN	**0.9900**	-
[95]	ECG^s	SA	SNN	0.9463	**0.9285**
Our work	ECG^s	SA	SNN	0.8955	0.6832
Our work	ECG^s	SA	ANN	0.8739	0.4930

^{s}Single-lead ECG, $F1^M$ (Macro)

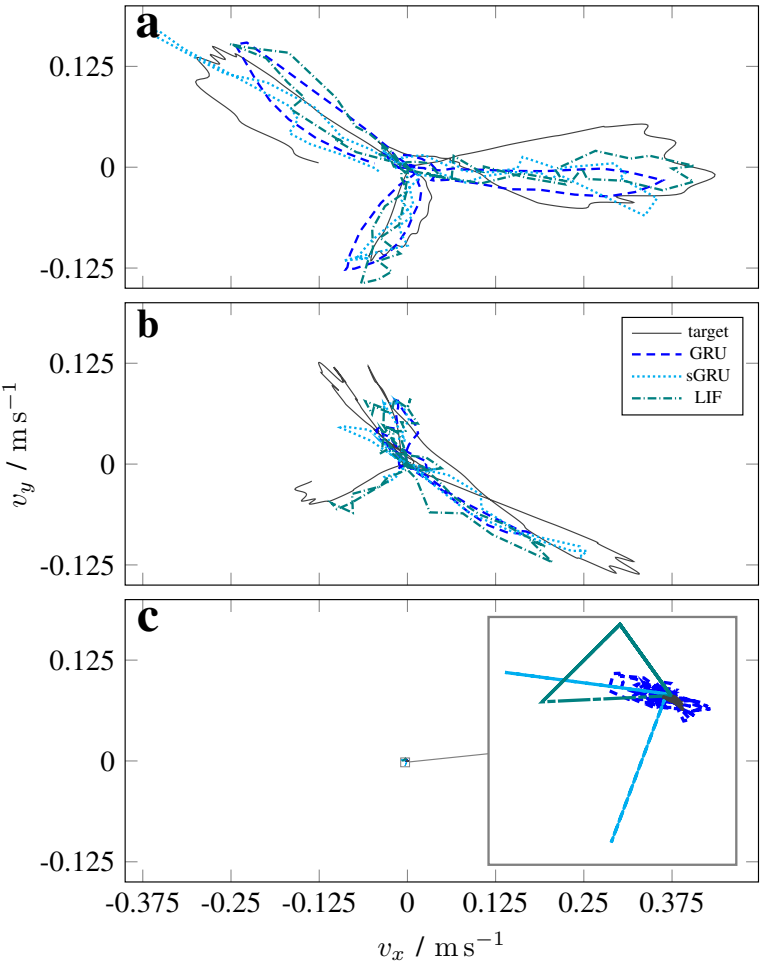


Figure A.1.: Visualization of the velocity outputs for the Track 2 model variants (GRU-t2, sGRU-t2, LIF-t2) for three exemplary test samples. **a**) High-accuracy reconstruction (R2 $\approx$ 0.9). **b**) Average accuracy (R2 $\approx$ 0.7). **c**) Low-accuracy reconstruction (R2 $\ll$ 0).

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial Neural Network
EA	Evolutionary Algorithm
CNN	Convolutional Neural Network
DCR	Deep Convolutional Recurrent
GRU	Gated Recurrent Unit
sGRU	Spiking Gated Recurrent Unit
WTA	Winner-Take-All
NAS	Neural Architecture Search
ReLU	Rectified Linear Unit
MLP	Multi-Layer Perceptron
LIF	Leaky Integrate-and-Fire
ALIF	Adaptive Leaky Integrate-and-Fire
CUBA	Current-Based
BLE	Bluetooth Low Energy
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
LSNN	Long Short-Term Spiking Neural Network
NN	Neural Network
SNN	Spiking Neural Network
DL	Deep Learning
FFT	Fast Fourier Transform
STFT	Short-Time Fourier transform
BPTT	Backpropagation Through Time
BP	Backpropagation
TTFS	Time-to-First-Spike
FPGA	Field Programmable Gate Array
SF	Step-Forward
SW	Sliding Window
BSA	Ben's Spiker Algorithm
PWM	Pulse Width Modulation

RL	Reinforcement Learning
FIR	Finite Impulse Response
MSE	Mean Squared Error
GRF	Gaussian Receptive Field
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MRPE	Mean Relative Percentage Error
FPR	False Positive Rate
FNR	False Negative Rate
AUC	Area Under the Curve
FSE	Full Scale Error
MVE	Measured Value Error
PM	Predictive Maintenance
PCP	Progressing Cavity Pump
MAC	Multiply-Accumulate Operation
ECG	Electrocardiogram
BMI	Brain-Machine Interface
iBMI	intra-cortical Brain-Machine Interface
OA	Obstructive Apnea
OH	Obstructive Hypopnea
STDP	Spike-Timing-Dependent Plasticity
LTP	Long-Term Potentiation
LTD	Long-Term Depression
R-STDP	Reward-modulated Spike-Timing-Dependent Plasticity
EONS	Evolutionary Optimization for Neuromorphic Systems
SNIP	Single-shot Network Pruning
SET	Sparse Evolutionary Training
GFB	Gammatone Filterbank
OSA	Obstructive Sleep Apnea
SOTA	State-of-the-Art

Bibliography

[1] *Access Intel Loihi Hardware - About the INRC - INRC.* https://intel-ncl.atlassian.net/wiki/spaces/INRC/pages/1810432001/Access+Intel+Loihi+Hardware

[2] *E2B-M12KN05-M1-B1 OMI | OMRON.* https://industrial.omron.de/de/products/E2B-M12KN05-M1-B1

[3] *NEMO® BY Blockpumpe in Industrieausführung.* https://pumps-systems.netzsch.com/de/produkte-und-zubehoer/nemo-exzenterschneckenpumpen/nemo-by-blockpumpe-in-industrieausfuehrung

[4] *PG2453 - Pressure sensor - ifm.* https://www.ifm.com/de/en/product/PG2453

[5] *Promag W 300, 5W3B1H.* https://www.de.endress.com/de/messgeraete-fuer-die-prozesstechnik/5W3B1H

[6] *SITRANS SCM IQ, SITRANS CC220, SITRANS MS200.* https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/7MP2210-2AB21-2AB1

[7] *farama-minari (Minari).* https://huggingface.co/farama-minari. Version: Februar 2025. – Accessed on 2025-06-05

[8] Achterberg, Jascha ; Akarca, Danyal ; Strouse, D. J. ; Duncan, John ; Astle, Duncan E.: Spatially embedded recurrent neural networks reveal widespread links between structural and functional neuroscience findings. In: *Nature Machine Intelligence* 5 (2023), November, Nr. 12, 1369–1381. http://dx.doi.org/10.1038/s42256-023-00748-9. – DOI 10.1038/s42256–023–00748–9. – ISSN 2522–5839

[9] Akiba, Takuya ; Sano, Shotaro ; Yanase, Toshihiko ; Ohta, Takeru ; Koyama, Masanori: Optuna: A next-generation hyperparameter optimization framework. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, S. 2623–2631

[10] Arriandiaga, Ander ; Portillo, Eva ; Espinosa-Ramos, Josafath I. ; Kasabov, Nikola K.: Pulsewidth modulation-based algorithm for spike phase encoding and decoding of time-dependent analog data. In: *IEEE Transactions on Neural Networks and Learning Systems* 31 (2019), Nr. 10, S. 3920–3931

[11] Azevedo, Frederico A. ; Carvalho, Ludmila R. ; Grinberg, Lea T. ; Farfel, José M. ; Ferretti, Renata E. ; Leite, Renata E. ; Filho, Wilson J. ; Lent, Roberto ; Herculano-Houzel, Suzana: Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. In: *Journal of Comparative Neurology* 513 (2009), Nr. 5, S. 532–541

[12] Balasubramanian, Vijay: Brain power. In: *Proceedings of the National Academy of Sciences* 118 (2021), Nr. 32, S. e2107022118

[13] Barchi, Francesco ; Zanatta, Luca ; Parisi, Emanuele ; Burrello, Alessio ; Brunelli, Davide ; Bartolini, Andrea ; Acquaviva, Andrea: Spiking Neural Network-Based Near-Sensor Computing for Damage Detection in Structural Health Monitoring. In: *Future Internet* 13 (2021), Nr. 8, 219. `http://dx.doi.org/10.3390/fi13080219`. – DOI 10.3390/fi13080219. – ISSN 1999–5903

[14] Bekolay, Trevor ; Bergstra, James ; Hunsberger, Eric ; DeWolf, Travis ; Stewart, Terrence ; Rasmussen, Daniel ; Choo, Xuan ; Voelker, Aaron ; Eliasmith, Chris: Nengo: a Python tool for building large-scale functional brain models. In: *Frontiers in Neuroinformatics* 7 (2014), Nr. 48, S. 1–13. `http://dx.doi.org/10.3389/fninf.2013.00048`. – DOI 10.3389/fninf.2013.00048. – ISSN 1662–5196

[15] Bellec, Guillaume ; Scherr, Franz ; Subramoney, Anand ; Hajek, Elias ; Salaj, Darjan ; Legenstein, Robert ; Maass, Wolfgang: A solution to the learning dilemma for recurrent networks of spiking neurons. In: *Nature communications* 11 (2020), Nr. 1, S.

3625

[16] Bogdan, Petruț A. ; Rowley, Andrew G. ; Rhodes, Oliver ; Furber, Steve B.: Structural plasticity on the spinnaker many-core neuromorphic system. In: *Frontiers in Neuroscience* 12 (2018), S. 434

[17] Chang, H.-Y. ; Yeh, C.-Y. ; Lee, C.-T. ; Lin, C.-C.: A sleep apnea detection system based on a one-dimensional deep convolution neural network model using single-lead electrocardiogram. In: *Sensors (Basel, Switzerland)* 20 (2020), Nr. 15, S. 4157. `http://dx.doi.org/10.3390/s20154157`. – DOI 10.3390/s20154157

[18] Chen, Quankun ; Li, Da ; Tao, Tuomin ; Ma, Hanzhi ; Li, Erping: Temporal Neural Encoding Methods for Spiking Neural Networks. In: *2022 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC)* IEEE, 2022, S. 88–90

[19] Davies, Mike ; Srinivasa, Narayan ; Lin, Tsung-Han ; Chinya, Gautham ; Cao, Yongqiang ; Choday, Sri H. ; Dimou, Georgios ; Joshi, Prasad ; Imam, Nabil ; Jain, Shweta u. a.: Loihi: A neuromorphic manycore processor with on-chip learning. In: *Ieee Micro* 38 (2018), Nr. 1, S. 82–99

[20] Davies, Mike ; Wild, Andreas ; Orchard, Garrick ; Sandamirskaya, Yulia ; Guerra, Gabriel A F. ; Joshi, Prasad ; Plank, Philipp ; Risbud, Sumedh R.: Advancing neuromorphic computing with loihi: A survey of results and outlook. In: *Proceedings of the IEEE* 109 (2021), Nr. 5, S. 911–934

[21] Deb, Mayukh ; Deb, Mainak ; Murty, N: TopoNets: High performing vision and language models with brain-like topography. In: *arXiv preprint arXiv:2501.16396* (2025)

[22] Degnan, Brian ; Marr, Bo ; Hasler, Jennifer: Assessing trends in performance per watt for signal processing applications. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24 (2015), Nr. 1, S. 58–66

[23] Dennler, Nik ; Haessig, Germain ; Cartiglia, Matteo ; Indiveri, Giacomo: Online Detection of Vibration Anomalies Using Balanced Spiking Neural Networks. In: *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*,

IEEE, 2021. – ISBN 978–1–66541–913–0, 1–4

[24] Dey, Sounak ; Banerjee, Dighanchal ; George, Arun M. ; Mukherjee, Arijit ; Pal, Arpan: Efficient Time Series Classification using Spiking Reservoir. In: *2022 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022. – ISBN 978–1–72818–671–9, 1–8

[25] Dupeyroux, Julien ; Stroobants, Stein ; De Croon, Guido C.: A toolbox for neuromorphic perception in robotics. In: *2022 8th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)* IEEE, 2022, S. 1–7

[26] Elbrecht, Daniel ; Schuman, Catherine: Neuroevolution of Spiking Neural Networks Using Compositional Pattern Producing Networks. In: *International Conference on Neuromorphic Systems 2020*. Oak Ridge TN USA : ACM, Juli 2020. – ISBN 978–1–4503–8851–1, S. 1–5

[27] Erdenebayar, U. ; Kim, Y. J. ; Park, J.-U. ; Joo, E. Y. ; Lee, K.-J.: Deep learning approaches for automatic detection of sleep apnea events from an electrocardiogram. In: *Computer Methods and Programs in Biomedicine* 180 (2019), S. 105001. `http://dx.doi.org/10.1016/j.cmpb.2019.105001`. – DOI 10.1016/j.cmpb.2019.105001

[28] Eshraghian, Jason K. ; Ward, Max ; Neftci, Emre ; Wang, Xinxin ; Lenz, Gregor ; Dwivedi, Girish ; Bennamoun, Mohammed ; Jeong, Doo S. ; Lu, Wei D.: Training spiking neural networks using lessons from deep learning. In: *Proceedings of the IEEE* 111 (2023), Nr. 9, S. 1016–1054

[29] Falcon, William ; The PyTorch Lightning team: *PyTorch Lightning*. `http://dx.doi.org/10.5281/zenodo.3828935`. Version: März 2019

[30] Frankle, Jonathan ; Carbin, Michael: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: *arXiv preprint arXiv:1803.03635* (2018)

[31] Georgiadis, Georgios: Accelerating convolutional neural networks via activation map compression. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, S. 7085–7095

[32] Gollahalli, Akshay R.: *github.com/akshaybabloo/Spikes*. `https://github.com/akshaybabloo/Spikes`. Version: September 2024. – original-date: 2016-10-05

[33] Hebb, Donald O.: *The organization of behavior: A neuropsychological theory*. Psychology press, 2005

[34] Hodgkin, A. L. ; Huxley, A. F.: A quantitative description of membrane current and its application to conduction and excitation in nerve. In: *The Journal of Physiology* 117 (1952), August, Nr. 4, 500–544. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413/`. – ISSN 0022–3751

[35] Hoefler, Torsten ; Alistarh, Dan ; Ben-Nun, Tal ; Dryden, Nikoli ; Peste, Alexandra: Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. In: *Journal of Machine Learning Research* 22 (2021), Nr. 241, S. 1–124

[36] Hofstadter, Douglas R.: *Gödel, Escher, Bach: an eternal golden braid*. 1979. – 360–371 S.

[37] Höppner, Sebastian ; Vogginger, Bernhard ; Yan, Yexin ; Dixius, Andreas ; Scholze, Stefan ; Partzsch, Johannes ; Neumärker, Felix ; Hartmann, Stephan ; Schiefer, Stefan ; Ellguth, Georg u. a.: Dynamic power management for neuromorphic many-core systems. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66 (2019), Nr. 8, S. 2973–2986

[38] Ito, Masao: Long-term depression. In: *Annual review of neuroscience* 12 (1989), S. 85–102

[39] Izhikevich, E.M.: Simple model of spiking neurons. In: *IEEE Transactions on Neural Networks* 14 (2003), November, Nr. 6, 1569–1572. `http://dx.doi.org/10.1109/TNN.2003.820440`. – DOI 10.1109/TNN.2003.820440. – ISSN 1941–0093

[40] Izhikevich, Eugene M.: Solving the distal reward problem through linkage of STDP and dopamine signaling. In: *Cerebral cortex* 17 (2007), Nr. 10, S. 2443–2452

[41] Jacobs, Robert A. ; Jordan, Michael I.: Computational Consequences of a Bias toward Short Connections. In: *Journal of Cognitive Neuroscience* 4 (1992), Oktober, Nr. 4, 323–

336. http://dx.doi.org/10.1162/jocn.1992.4.4.323. – DOI 10.1162/jocn.1992.4.4.323. – ISSN 0898–929X

[42] Jia, Z. ; Ji, J. ; Zhou, X. ; Zhou, Y.: Hybrid spiking neural network for sleep electroencephalogram signals. In: *Science China Information Sciences* 65 (2022), Nr. 4. http://dx.doi.org/10.1007/s11432-021-3380-1. – DOI 10.1007/s11432–021–3380–1

[43] Kasabov, Nikola ; Scott, Nathan M. ; Tu, Enmei ; Marks, Stefan ; Sengupta, Neelava ; Capecci, Elisa ; Othman, Muhaini ; Doborjeh, Maryam G. ; Murli, Norhanifah ; Hartono, Reggio u. a.: Evolving spatio-temporal data machines based on the NeuCube neuromorphic framework: Design methodology and selected applications. In: *Neural Networks* 78 (2016), S. 1–14

[44] Khalighi, S. ; Sousa, T. ; Santos, J. M. ; Nunes, U.: ISRUC-Sleep: A comprehensive public dataset for sleep researchers. In: *Computer Methods and Programs in Biomedicine* 124 (2016), S. 180–192. http://dx.doi.org/10.1016/j.cmpb.2015.10.013. – DOI 10.1016/j.cmpb.2015.10.013

[45] Kholkin, Vladislav ; Druzhina, Olga ; Vatnik, Valerii ; Kulagin, Maksim ; Karimov, Timur ; Butusov, Denis: Comparing Reservoir Artificial and Spiking Neural Networks in Machine Fault Detection Tasks. In: *BDCC* 7 (2023), Nr. 2, 110. http://dx.doi.org/10.3390/bdcc7020110. – DOI 10.3390/bdcc7020110. – ISSN 2504–2289

[46] Kingma, Diederik P. ; Ba, Jimmy: Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980* (2014)

[47] Kurtz, Mark ; Kopinsky, Justin ; Gelashvili, Rati ; Matveev, Alexander ; Carr, John ; Goin, Michael ; Leiserson, William ; Moore, Sage ; Shavit, Nir ; Alistarh, Dan: Inducing and exploiting activation sparsity for fast inference on deep neural networks. In: *International Conference on Machine Learning* PMLR, 2020, S. 5533–5543

[48] Kurtzer, Isaac L.: Long-latency reflexes account for limb biomechanics through several supraspinal pathways. In: *Frontiers in Integrative Neuroscience* 8 (2015), Januar. http://dx.doi.org/10.3389/fnint.2014.00099. – DOI 10.3389/fnint.2014.00099. – ISSN 1662–5145. – Publisher: Frontiers

[49] Lebedev, Mikhail: Brain-machine interfaces: an overview. In: *Translational Neuroscience* 5 (2014), S. 99–110

[50] LeCun, Yann ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nr. 11, S. 2278–2323. `http://dx.doi.org/10.1109/5.726791`. – DOI 10.1109/5.726791

[51] Lee, Namhoon ; Ajanthan, Thalaiyasingam ; Torr, Philip H.: Snip: Single-shot network pruning based on connection sensitivity. In: *arXiv preprint arXiv:1810.02340* (2018)

[52] Lennie, Peter: The cost of cortical computation. In: *Current biology* 13 (2003), Nr. 6, S. 493–497

[53] Li, Hao ; Kadav, Asim ; Durdanovic, Igor ; Samet, Hanan ; Graf, Hans P.: Pruning filters for efficient convnets. In: *arXiv preprint arXiv:1608.08710* (2016)

[54] Libedinsky, Camilo ; So, Rosa ; Xu, Zhiming ; Kyar, Toe K. ; Ho, Duncun ; Lim, Clement ; Chan, Louiza ; Chua, Yuanwei ; Yao, Lei ; Cheong, Jia H. u. a.: Independent mobility achieved through a wireless brain-machine interface. In: *PLoS One* 11 (2016), Nr. 11, S. e0165773

[55] Ling, V. ; Wu, C. ; Stiles, S.: Sleep apnea statistics and facts you should know. In: *NCOA Adviser* (2023), October. `https://www.ncoa.org/adviser/sleep/sleep-apnea-statistics/`

[56] Liu, Tengjun ; Gygax, Julia ; Rossbroich, Julian ; Chua, Yansong ; Zhang, Shaomin ; Zenke, Friedemann: Decoding finger velocity from cortical spike trains with recurrent spiking neural networks. (2024), S. 1–5

[57] Loparo, KA: Case western reserve university bearing data center. In: *Bearings Vibration Data Sets, Case Western Reserve University* (2012), S. 22–28

[58] Maass, Wolfgang: Networks of spiking neurons: the third generation of neural network models. In: *Neural networks* 10 (1997), Nr. 9, S. 1659–1671

[59] Makowski, Dominique ; Pham, Tam ; Lau, Zen J. ; Brammer, Jan C. ; Lespinasse, François ; Pham, Hung ; Schölzel, Christo-

pher ; Chen, S. H. A.: NeuroKit2: A Python toolbox for neurophysiological signal processing. In: *Behavior Research Methods* 53 (2021), feb, Nr. 4, 1689–1696. http://dx.doi.org/10.3758/s13428-020-01516-y. – DOI 10.3758/s13428–020–01516–y

[60] Mashrur, F. R. ; Islam, M. S. ; Saha, D. K. ; Islam, S. M. R. ; Moni, M. A.: SCNN: Scalogram-based convolutional neural network to detect obstructive sleep apnea using single-lead electrocardiogram signals. In: *Computers in Biology and Medicine* 134 (2021), S. 104532. http://dx.doi.org/10.1016/j.compbiomed.2021.104532. – DOI 10.1016/j.compbiomed.2021.104532

[61] Mead, Carver: *Analog VLSI and neural systems*. USA : Addison-Wesley Longman Publishing Co., Inc., 1989. – ISBN 0201059924

[62] Meister, Markus ; Berry, Michael J.: The neural code of the retina. In: *neuron* 22 (1999), Nr. 3, S. 435–450

[63] Merolla, Paul A. ; Arthur, John V. ; Alvarez-Icaza, Rodrigo ; Cassidy, Andrew S. ; Sawada, Jun ; Akopyan, Filipp ; Jackson, Bryan L. ; Imam, Nabil ; Guo, Chen ; Nakamura, Yutaka u. a.: A million spiking-neuron integrated circuit with a scalable communication network and interface. In: *Science* 345 (2014), Nr. 6197, S. 668–673

[64] Mocanu, Decebal C. ; Mocanu, Elena ; Stone, Peter ; Nguyen, Phuong H. ; Gibescu, Madeleine ; Liotta, Antonio: Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. In: *Nature communications* 9 (2018), Nr. 1, S. 2383

[65] Morrison, Abigail ; Aertsen, Ad ; Diesmann, Markus: Spike-Timing-Dependent Plasticity in Balanced Random Networks. In: *Neural Computation* 19 (2007), 06, Nr. 6, 1437-1467. http://dx.doi.org/10.1162/neco.2007.19.6.1437. – DOI 10.1162/neco.2007.19.6.1437. – ISSN 0899–7667

[66] Mukherjee, D. ; Dhar, K. ; Schwenker, F. ; Sarkar, R.: Ensemble of deep learning models for sleep apnea detection: An experimental study. In: *Sensors (Basel, Switzerland)* 21 (2021), Nr. 16, S. 5425. http://dx.doi.org/10.3390/s21165425. – DOI 10.3390/s21165425

[67] Nasifoglu, H. ; Erogul, O.: Obstructive sleep apnea prediction from electrocardiogram scalograms and spectrograms using convolutional neural networks. In: *Physiological Measurement* 42 (2021), Nr. 6, S. 065010. http://dx.doi.org/10.1088/1361-6579/ac0a9c. – DOI 10.1088/1361–6579/ac0a9c

[68] Neftci, Emre O. ; Mostafa, Hesham ; Zenke, Friedemann: Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. In: *IEEE Signal Processing Magazine* 36 (2019), Nr. 6, S. 51–63

[69] Nitzsche, Sven ; Neher, Moritz ; Dosky, Stefan von ; Becker, Jürgen: Ultra-low Power Machinery Fault Detection Using Deep Neural Networks. In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Springer International Publishing, 2021 (Communications in Computer and Information Science). – ISBN 978–3–030–93736–2, S. 390–396

[70] O'Doherty, Joseph E. ; Cardoso, Mariana M. ; Makin, Joseph G. ; Sabes, Philip N.: Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology. In: *Zenodo http://doi.org/10.5281/zenodo* 583331 (2017)

[71] Olshausen, Bruno A. ; Field, David J.: Sparse coding of sensory inputs. In: *Current opinion in neurobiology* 14 (2004), Nr. 4, S. 481–487

[72] Ozaki, Yoshihiko ; Tanigaki, Yuki ; Watanabe, Shuhei ; Onishi, Masaki: Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In: *Proceedings of the 2020 genetic and evolutionary computation conference*, 2020, S. 533–541

[73] Paszke, Adam ; Gross, Sam ; Massa, Francisco ; Lerer, Adam ; Bradbury, James ; Chanan, Gregory ; Killeen, Trevor ; Lin, Zeming ; Gimelshein, Natalia ; Antiga, Luca ; Desmaison, Alban ; Kopf, Andreas ; Yang, Edward ; DeVito, Zachary ; Raison, Martin ; Tejani, Alykhan ; Chilamkurthy, Sasank ; Steiner, Benoit ; Fang, Lu ; Bai, Junjie ; Chintala, Soumith: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32*. Curran

Associates, Inc., 2019, S. 8024–8035

[74] PATTERSON, R.: An efficient auditory filterbank based on the gammatone function. In: *Meeting of the IOC Speech Group on Auditory Modelling at RSRE, 1987* (1987). https://cir.nii.ac.jp/crid/1570291225803648384

[75] Pehle, Christian ; Billaudelle, Sebastian ; Cramer, Benjamin ; Kaiser, Jakob ; Schreiber, Korbinian ; Stradmann, Yannik ; Weis, Johannes ; Leibfried, Aron ; Müller, Eric ; Schemmel, Johannes: The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity. In: *Frontiers in Neuroscience* 16 (2022), S. 795876

[76] Pehle, Christian-Gernot ; Egholm Pedersen, Jens: Norse-A deep learning library for spiking neural networks. In: *Zenodo* (2021)

[77] Petro, Balint ; Kasabov, Nikola ; Kiss, Rita M.: Selection and optimization of temporal spike encoding methods for spiking neural networks. In: *IEEE transactions on neural networks and learning systems* 31 (2019), Nr. 2, S. 358–370

[78] Pini, N. ; Ong, J. L. ; Yilmaz, G. ; Chee, N. I. Y. N. ; Siting, Z. ; Awasthi, A. ; Biju, S. ; Kishan, K. ; Patanaik, A. ; Fifer, W. P. ; Lucchini, M.: An automated heart rate-based algorithm for sleep stage classification: Validation using conventional polysomnography and an innovative wearable electrocardiogram device. In: *Frontiers in Neuroscience* 16 (2022), S. 974192. http://dx.doi.org/10.3389/fnins.2022.974192. – DOI 10.3389/fnins.2022.974192

[79] Qiu, Huanneng ; Garratt, Matthew ; Howard, David ; Anavatti, Sreenatha: Evolving Spiking Neural Networks for Nonlinear Control Problems. In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2018, S. 1367–1373. – arXiv:1903.01180 [cs]

[80] Reichert, William M.: *Indwelling neural implants: strategies for contending with the in vivo environment*. CRC Press, 2007

[81] Roy, Kaushik ; Jaiswal, Akhilesh ; Panda, Priyadarshini: Towards spike-based machine intelligence with neuromorphic computing. In: *Nature* 575 (2019), Nr. 7784, S. 607–617

[82] Rueckauer, Bodo ; Lungu, Iulia-Alexandra ; Hu, Yuhuang ; Pfeiffer, Michael ; Liu, Shih-Chii: Conversion of continuous-

valued deep networks to efficient event-driven networks for image classification. In: *Frontiers in neuroscience* 11 (2017), S. 682

[83] Sakemi, Yusuke ; Yamamoto, Kakei ; Hosomi, Takeo ; Aihara, Kazuyuki: Sparse-firing regularization methods for spiking neural networks with time-to-first-spike coding. In: *Scientific Reports* 13 (2023), Nr. 1, S. 22897

[84] Schrauwen, Benjamin ; Van Campenhout, Jan: BSA, a fast and accurate spike train encoding scheme. In: *Proceedings of the International Joint Conference on Neural Networks, 2003.* Bd. 4 IEEE, 2003, S. 2825–2830

[85] Schulman, John ; Wolski, Filip ; Dhariwal, Prafulla ; Radford, Alec ; Klimov, Oleg: Proximal Policy Optimization Algorithms. In: *CoRR* abs/1707.06347 (2017)

[86] Schuman, Catherine D. ; Mitchell, J P. ; Patton, Robert M. ; Potok, Thomas E. ; Plank, James S.: Evolutionary optimization for neuromorphic systems. In: *Proceedings of the 2020 Annual Neuro-Inspired Computational Elements Workshop*, 2020, S. 1–9

[87] Shanechi, Maryam M. ; Orsborn, Amy L. ; Moorman, Helene G. ; Gowda, Suraj ; Dangi, Siddharth ; Carmena, Jose M.: Rapid control and feedback rates enhance neuroprosthetic control. In: *Nature communications* 8 (2017), Nr. 1, S. 13825

[88] Shannon, Claude E.: A mathematical theory of communication. In: *The Bell system technical journal* 27 (1948), Nr. 3, S. 379–423

[89] Shrestha, Sumit B. ; Orchard, Garrick: Slayer: Spike layer error reassignment in time. In: *Advances in neural information processing systems* 31 (2018)

[90] Simeral, John D. ; Hosman, Thomas ; Saab, Jad ; Flesher, Sharlene N. ; Vilela, Marco ; Franco, Brian ; Kelemen, Jessica N. ; Brandman, David M. ; Ciancibello, John G. ; Rezaii, Paymon G. u. a.: Home use of a percutaneous wireless intracortical brain-computer interface by individuals with tetraplegia. In: *IEEE Transactions on Biomedical Engineering* 68 (2021), Nr. 7, S. 2313–2325

[91] Stiso, Jennifer ; Bassett, Danielle: *Spatial Embedding Imposes Constraints on the Network Architectures of Neural Systems*. http://arxiv.org/abs/1807.04691. Version: Juli 2018. – arXiv:1807.04691 [q-bio]

[92] Sun, C. ; Hong, S. ; Wang, J. ; Dong, X. ; Han, F. ; Li, H.: A systematic review of deep learning methods for modeling electrocardiograms during sleep. In: *Physiological Measurement* 43 (2022), Nr. 8, S. 08TR02. http://dx.doi.org/10.1088/1361-6579/ac826e. – DOI 10.1088/1361–6579/ac826e

[93] Todorov, Emanuel ; Erez, Tom ; Tassa, Yuval: Mujoco: A physics engine for model-based control. In: *2012 IEEE/RSJ international conference on intelligent robots and systems* IEEE, 2012, S. 5026–5033

[94] Towers, Mark ; Terry, Jordan K. ; Kwiatkowski, Ariel ; Balis, John U. ; Cola, Gianluca d. ; Deleu, Tristan ; Goulão, Manuel ; Kallinteris, Andreas ; KG, Arjun ; Krimmel, Markus ; Perez-Vicente, Rodrigo ; Pierré, Andrea ; Schulhoff, Sander ; Tai, Jun J. ; Shen, Andrew Tan J. ; Younis, Omar G.: *Gymnasium*. http://dx.doi.org/10.5281/zenodo.8127026. Version: März 2023

[95] Tyagi, P. K. ; Agrawal, D.: Automatic detection of sleep apnea from a single-lead ECG signal based on spiking neural network model. In: *Computers in Biology and Medicine* 179 (2024), S. 108877. http://dx.doi.org/10.1016/j.compbiomed.2024.108877. – DOI 10.1016/j.compbiomed.2024.108877

[96] Urtnasan, E. ; Park, J.-U. ; Joo, E. Y. ; Lee, K.-J.: Deep convolutional recurrent model for automatic scoring sleep stages based on single-lead ECG signal. In: *Diagnostics (Basel, Switzerland)* 12 (2022), Nr. 5, S. 1235. http://dx.doi.org/10.3390/diagnostics12051235. – DOI 10.3390/diagnostics12051235

[97] Vasilache, Alexandru: *Alex-Vasilache/Spike-Encoding*. Februar 2025. – [Online]. Available: https://github.com/Alex-Vasilache/Spike-Encoding

[98] Vishwakarma, Manish ; Purohit, Rajesh ; Harshlata, V ; Rajput, PRAMOD: Vibration analysis and condition monitoring

for rotating machines: a review. In: *Materials Today: Proceedings* 4 (2017), Nr. 2, S. 2659–2664

[99] Wang, Bo ; Ke, Wei ; Guang, Jing ; Chen, Guang ; Yin, Luping ; Deng, Suixin ; He, Quansheng ; Liu, Yaping ; He, Ting ; Zheng, Rui u. a.: Firing frequency maxima of fast-spiking neurons in human, monkey, and mouse neocortex. In: *Frontiers in cellular neuroscience* 10 (2016), S. 239

[100] Wang, J.-S. ; Shih, G.-R. ; Chiang, W.-C.: Sleep stage classification of sleep apnea patients using decision-tree-based support vector machines based on ECG parameters. In: *Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics*, 2012

[101] Wang, Kuanchuan ; Hao, Xinyu ; Wang, Jiang ; Deng, Bin: Comparison and Selection of Spike Encoding Algorithms for SNN on FPGA. In: *IEEE Transactions on Biomedical Circuits and Systems* 17 (2023), Nr. 1, S. 129–141

[102] Wang, Yuanxi ; Wang, Zuowen ; Liu, Shih-Chii: Leveraging recurrent neural networks for predicting motor movements from primate motor cortex neural recordings. (2024), S. 1–5

[103] Webb, Andrew ; Davies, Sergio ; Lester, David: Spiking neural PID controllers. In: *Neural Information Processing: 18th International Conference, ICONIP 2011, Shanghai, China, November 13-17, 2011, Proceedings, Part III 18* Springer, 2011, S. 259–267

[104] Wei, Y. ; Qi, X. ; Wang, H. ; Liu, Z. ; Wang, G. ; Yan, X.: A multi-class automatic sleep staging method based on long short-term memory network using single-lead electrocardiogram signals. In: *IEEE Access: Practical Innovations, Open Solutions* 7 (2019), S. 85959–85970. `http://dx.doi.org/10.1109/access.2019.2924980`. – DOI 10.1109/access.2019.2924980

[105] Wunderlich, Timo C. ; Pehle, Christian: Event-based backpropagation can compute exact gradients for spiking neural networks. In: *Scientific Reports* 11 (2021), Nr. 1, S. 12829

[106] Xie, Xiaohui ; Hahnloser, Richard ; Seung, H S.: Learning winner-take-all competition between groups of neurons in lateral inhibitory networks. In: *Advances in neural information processing*

systems 13 (2000)

[107] Yarga, Sidi Yaya A. ; Rouat, Jean ; Wood, Sean: Efficient spike encoding algorithms for neuromorphic speech recognition. In: *Proceedings of the International Conference on Neuromorphic Systems 2022*, 2022, S. 1–8

[108] Yik, Jason ; Ahmed, Soikat H. ; Ahmed, Zergham ; Anderson, Brian ; Andreou, Andreas G. ; Bartolozzi, Chiara ; Basu, Arindam ; Blanken, Douwe den ; Bogdan, Petrut ; Buckley, Sonia u. a.: Neurobench: Advancing neuromorphic computing through collaborative, fair and representative benchmarking. (2023)

[109] Yu, Xiaolong ; Tian, Cong: Dual sparse training framework: inducing activation map sparsity via Transformed ℓ_1 regularization. In: *arXiv preprint arXiv:2405.19652* (2024)

[110] Yuan, Kun ; Li, Quanquan ; Shao, Jing ; Yan, Junjie: Learning connectivity of neural networks from a topological perspective. In: *European Conference on Computer Vision* Springer, 2020, S. 737–753

[111] Yücelbaş, Şule ; Yücelbaş, Cüneyt ; Tezel, Gülay ; Özşen, Seral ; Yosunkaya, Şebnem: Automatic sleep staging based on SVD, VMD, HHT and morphological features of single-lead ECG signal. In: *Expert Systems with Applications* 102 (2018), S. 193–206

[112] Zanatta, Luca ; Barchi, Francesco ; Burrello, Alessio ; Bartolini, Andrea ; Brunelli, Davide ; Acquaviva, Andrea: Damage Detection in Structural Health Monitoring with Spiking Neural Networks. In: *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, IEEE, 2021. – ISBN 978–1–66541–980–2, 105–110

[113] Zanatta, Luca ; Barchi, Francesco ; Manoni, Simone ; Tolu, Silvia ; Bartolini, Andrea ; Acquaviva, Andrea: *Exploring Spiking Neural Networks for Deep Reinforcement Learning in Robotic Tasks: A Comparative Study*

[114] Zenke, Friedemann ; Ganguli, Surya: Superspike: Supervised learning in multilayer spiking neural networks. In: *Neural computation* 30 (2018), Nr. 6, S. 1514–1541

[115] Zhang, J. ; Tang, Z. ; Gao, J. ; Lin, L. ; Liu, Z. ; Wu, H. ; Liu, F. ; Yao, R.: Automatic detection of obstructive sleep apnea events using a deep CNN-LSTM model. In: *Computational Intelligence and Neuroscience* 2021 (2021), S. 5594733. `http://dx.doi.org/10.1155/2021/5594733`. – DOI 10.1155/2021/5594733

[116] Zhang, Jiawei: Basic neural units of the brain: neurons, synapses and action potential. In: *arXiv preprint arXiv:1906.01703* (2019)

[117] Zhang, Xiayin ; Ma, Ziyue ; Zheng, Huaijin ; Li, Tongkeng ; Chen, Kexin ; Wang, Xun ; Liu, Chenting ; Xu, Linxi ; Wu, Xiaohang ; Lin, Duoru u. a.: The combination of brain-computer interfaces and artificial intelligence: applications and challenges. In: *Annals of translational medicine* 8 (2020), Nr. 11

Supervised Student Works

[Erb25] Erb, Laura: *Training Neural Networks Through Topology Optimization*. Karlsruhe, Karlsruhe Institute of Technology (KIT), Bachelor's Thesis, April 2025

[Jan23] Janocha, Magdalena: *Analyzing Converter Selection for Robotic Applications with Spiking Neural Networks*, Karlsruhe Institute of Technology (KIT), Seminar Thesis, November 2023

[Neu23] Neumann, Oliver: *Analyse von Codierungsalgorithmen für Spiking Neuronal Networks (SNN) in Anwendung auf EKG-Signale*, Karlsruhe Institute of Technology (KIT), Seminar Thesis, November 2023

[Sch24a] Schilling, Vincent: *Evaluation of Embedded Spike Encoding Algorithms*, Karlsruhe Institute of Technology (KIT), Bachelor's Thesis, September 2024

[Sch24b] Schilling, Vincent: *Spiking Neural Networks for Multi-Agent Collaborative Tasks*, Karlsruhe Institute of Technology (KIT), Seminar Thesis, März 2024

[Sch25] Scholz, Jona: *Solving Classic Control Problems with Actor-Critic Spiking Neural Networks*, FernUniversität in Hagen, Master's Thesis, Januar 2025

[Sef24] Sefrin, Max: *Evolutionäre Algorithmen mit Spiking Neural Networks in der Robotik*, Karlsruhe Institute of Technology (KIT), Seminar Thesis, März 2024

[Tho24] Thoemmes, Felix: *Sensor Pen Trajectory Reconstruction Using Spiking Neural Networks*, Karlsruhe Institute of Technology (KIT), Master's Thesis, April 2024

[Tre25] Tretter, Florian: *Neuromorphic Motion Planning in Pedestrian-rich Environments using Reinforcement Learning*

with Spiking Neural Networks, Master's Thesis, Mai 2025

[Tru23] TRUMPA, Adrian: *Spiking Neural Networks - Encoding und Decoding von Vibrationsdaten*, Karlsruhe Institute of Technology (KIT), Seminar Thesis, November 2023

Conference Contributions

[BVH+24] Biri*, Gergely I. ; Vasilache*, Alexandru ; Hu, Tao ; Themistocli, Maximilian Alexander N. ; Nitzsche, Sven ; Juhl, Jens ; Erler, Christina ; Fuhrhop, Silvester ; Stork, Wilhelm ; Becker, Juergen: Sleep Stage and Apnea Classification from Single-Lead ECG Using Artificial and Spiking Neural Networks. In: *2024 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*. Penang, Malaysia, Dezember 2024, 79–84. – *Equal contribution, Best Paper Award, ISSN: 2573-3028

[EBV+25] Erb, Laura ; Boccato, Tommaso ; Vasilache, Alexandru ; Becker, Juergen ; Toschi, Nicola: Training Neural Networks by Optimizing Neuron Positions. In: *2025 LIVING MACHINES*. Sheffield, UK, Juli 2025

[KBB+25] Knapp, Leonard ; Borsig, Matthias ; Baumgart, Ingmar ; Nitzsche, Sven ; Vasilache, Alexandru ; Becker, Juergen: Efficacy of Spiking Neural Networks for Intrusion Detection Systems. In: *2025 The International Conference on Cybersecurity and AI-Based Systems (Cyber-AI)*. Varna, Bulgaria, September 2025

[KVKB25] Krausse*, Jann ; Vasilache*, Alexandru ; Knobloch, Klaus ; Becker, Juergen: Realtime-Capable Hybrid Spiking Neural Networks for Neural Decoding of Cortical Activity. In: *2025 Neuro Inspired Computational Elements Conference (NICE)*. Heidelberg, Germany, März 2025. – *Equal contribution

[VKKB24] Vasilache*, Alexandru ; Krausse*, Jann ; Knobloch, Klaus ; Becker, Juergen: Hybrid Spiking Neural Networks for Low-Power Intra-Cortical Brain-Machine Interfaces. In: *2024*

IEEE Biomedical Circuits and Systems Conference (BioCAS). Xi'an, China, Oktober 2024, 1–5. – *Equal contribution, 2nd place winner in the Grand Challenge on Neural Decoding, ISSN: 2766-4465

[VNF+24] Vasilache, Alexandru ; Nitzsche, Sven ; Floegel, Daniel ; Schuermann, Tobias ; Bierweiler, Thomas ; Mussler, Marvin ; Kalber, Florian ; Hohmann, Soeren ; Becker, Juergen: Low-Power Vibration-Based Predictive Maintenance for Industry 4.0 using Neural Networks: A Survey. In: *2024 Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECMLPKDD)*. Vilnius, Lithuania, September 2024

[VNK+25] Vasilache, Alexandru ; Nitzsche, Sven ; Kneidl, Christian ; Tekneyan, Mikael ; Neher, Moritz ; Becker, Juergen: Spiking Neural Networks for Low-Power Vibration-Based Predictive Maintenance. In: *2025 International Conference on Neuromorphic Systems (ICONS)*. Seattle, US, Juli 2025

[VSS+25] Vasilache, Alexandru ; Scholz, Jona ; Schilling, Vincent ; Nitzsche, Sven ; Kaelber, Florian ; Korsch, Johannes ; Becker, Juergen: A PyTorch-Compatible Spike Encoding Framework for Energy-Efficient Neuromorphic Applications. In: *2025 International Conference on Systems (ICONS)*. Nice, France, Mai 2025. – Best Paper Award

[VSSB25] Vasilache, Alexandru ; Scholz, Jona ; Sandamirskaya, Yulia ; Becker, Juergen: Evolving Spatially Embedded Recurrent Spiking Neural Networks for Control Tasks. In: *2025 International Conference on Artificial Neural Networks (ICANN)*. Kaunas, Lithuania, September 2025. – Best Student Paper Award