

# Adaptively Pruned Spiking Neural Networks for Energy-Efficient Intracortical Neural Decoding

Francesca Rivelli, Martin Popov, Charalampos S. Kouzinopoulos and Guangzhi Tang

**Abstract**—Intracortical brain-machine interfaces demand low-latency, energy-efficient solutions for neural decoding. Spiking Neural Networks (SNNs) deployed on neuromorphic hardware have demonstrated remarkable efficiency in neural decoding by leveraging sparse binary activations and efficient spatiotemporal processing. However, reducing the computational cost of SNNs remains a critical challenge for developing ultra-efficient intracortical neural implants. In this work, we introduce a novel adaptive pruning algorithm specifically designed for SNNs with high activation sparsity, targeting intracortical neural decoding. Our method dynamically adjusts pruning decisions and employs a rollback mechanism to selectively eliminate redundant synaptic connections without compromising decoding accuracy. Experimental evaluation on the NeuroBench Non-Human Primate (NHP) Motor Prediction benchmark shows that our pruned network achieves performance comparable to dense networks, with a maximum tenfold improvement in efficiency. Moreover, hardware simulation on the neuromorphic processor reveals that the pruned network operates at sub- $\mu\text{W}$  power levels, underscoring its potential for energy-constrained neural implants. These results underscore the promise of our approach for advancing energy-efficient intracortical brain-machine interfaces with low-overhead on-device intelligence.

## I. INTRODUCTION

Intracortical neural implants play a pivotal role in brain-machine interfaces (BMIs) and neuroprosthetic systems by enabling the restoration of lost sensory, motor, or cognitive functions [1], [2]. For these devices to operate effectively, they must process neural signals in real-time within the brain under strict power constraints imposed by battery or wireless energy transfer [3], [4]. Low-latency processing is essential to ensure immediate response, which is critical for applications such as prosthetic limb control and speech decoding [5]. Furthermore, heat dissipation is a critical concern limiting the maximum power consumption, because even minor temperature increases in the sensitive environment of the human brain can jeopardize surrounding tissues [6]. Recent advances in deep learning have significantly improved the accuracy of intracortical neural decoding [7], [8]. However, the substantial computational cost associated with deep neural networks hinders their direct implementation

Francesca Rivelli, Martin Popov, Charalampos S. Kouzinopoulos and Guangzhi Tang are with the Department of Advanced Computing Sciences, Faculty of Science and Engineering, Maastricht University, Maastricht, The Netherlands. [guangzhi.tang@maastrichtuniversity.nl](mailto:guangzhi.tang@maastrichtuniversity.nl)

This publication is part of the project *Brain-inspired MatMul-free Deep Learning for Sustainable AI on Neuromorphic Processor* with file number NGF.1609.243.044 of the research programme AiNed XS Europe which is (partly) financed by the Dutch Research Council (NWO) under the grant <https://doi.org/10.61686/MYMX53467>.

in resource-constrained intracortical neural implants, thereby necessitating the development of more efficient solutions.

Neuromorphic computing using Spiking Neural Networks (SNNs) has demonstrated significant energy efficiency improvements across a broad spectrum of applications [9], [10], [11], [12]. In contrast to conventional deep neural networks, SNNs process spatiotemporal information efficiently by leveraging sparse, event-driven computations and stateful spiking neurons [13]. SNN-based approaches for intracortical neural decoding have been investigated and have demonstrated performance comparable to state-of-the-art methods, while incurring lower energy costs [14]. Furthermore, the collaborative NeuroBench benchmark for neuromorphic computing has designated intracortical neural decoding as one of its initial tasks, underscoring the significance of this application in demonstrating the advantages of neuromorphic systems [15]. Although SNNs have already demonstrated efficient neural decoding, an open challenge remains in further reducing computational costs to meet the stringent power constraints, ultimately at the sub- $\mu\text{W}$  level [16], required for ultra-efficient intracortical neural implants.

Synaptic pruning reduces the computational cost of neural networks by eliminating unnecessary neurons and synaptic connections [17]. Various pruning strategies employ metrics such as synaptic weight magnitudes [18], changes in loss [19], and activation patterns [20] to systematically remove network components that contribute minimally to overall performance. In SNNs, pruning not only reduces the number of weights but also decreases the number of synaptic operations per inference, thereby lowering overall computational costs. For instance, weight magnitude-based pruning [21] and activity-based pruning [22] are two methods that target distinct SNN characteristics to enhance efficiency. However, additional challenges arise when SNNs exhibit extremely high activation sparsity during neural decoding. In such cases, the network becomes highly sensitive to synaptic pruning, and conventional pruning methods can lead to significant performance degradation.

In this paper, we propose an adaptive pruning algorithm to further enhance the energy efficiency of SNNs for intracortical neural decoding. Our algorithm accelerates both the pruning speed and overall pruning rate by incorporating an adaptive execution strategy, and it directly addresses the sensitive pruning problem through an adaptive rollback mechanism during the pruning process. We evaluated our approach using the NeuroBench Non-Human Primate (NHP) Motor Prediction benchmark [15]. Compared with dense networks of similar architectures, SNNs pruned by our

adaptive pruning algorithm achieve comparable prediction performance while delivering over a maximum  $10\times$  improvement in efficiency. Furthermore, a hardware simulation study employing realistic measurements from the SENECA neuromorphic processor [23] demonstrates that our pruned SNN can achieve sub- $\mu$ W power consumption, thereby significantly reducing the computational overhead of neural decoding in intracortical neural implants.

## II. METHOD

We employ Spiking Neural Networks (SNNs) with stateful Leaky Integrate-and-Fire (LIF) neural models to decode spatiotemporal information from intracortical neural signals recorded during the Primate Reaching task. To improve energy efficiency and reduce computational and memory requirements, we developed an adaptive pruning algorithm that minimizes the number of synapses and synaptic operations without compromising the performance of the SNN.

### A. Spiking Neural Network

In this study, we employ a multilayer SNN, illustrated in Figure 1, to decode neural signals recorded during the Primate Reaching task. The architecture, referred to as SNN3 in [14], comprises three hidden layers and one output layer. All hidden layers consist of spiking LIF neurons, while the output layer comprises two non-spiking LIF neurons whose membrane potentials encode the two-dimensional finger velocity. The inputs are binary spikes from thresholding the intracortical neural recordings. The input dimension varies by session, with 96 channels for Indy sessions and 192 channels for Loco sessions, reflecting differences in the recording devices. The fully connected SNN contains 9,900 synapses for Indy sessions and 14,700 synapses for Loco sessions. We applied our adaptive pruning algorithm to reduce the number of synapses (weights) in the hidden layers of the network.

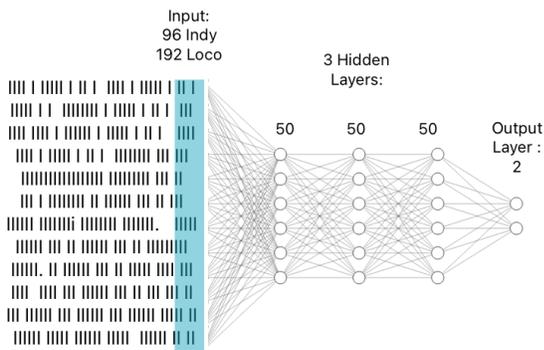


Fig. 1. Spiking neural network with 3 hidden layers for decoding recorded neural signals of the Primate Reaching task. The input, hidden, and output dimensions are listed. The adaptive pruning algorithm is applied to the hidden layers of the network.

### B. Leaky Integrate-and-Fire (LIF) Neurons

In our SNN, stateful LIF neurons are employed to capture the spatiotemporal dynamics of the recorded neural signals.

At each timestep, each LIF neuron updates its state through two sequential processes [24]: first, the membrane potential is updated, and then the neuron may generate a spike. In the hidden layers, neurons execute both the membrane potential update and the spiking process, whereas neurons in the output layer update their membrane potentials without generating spikes.

1) *Membrane potential update*: The membrane potential of a neuron at a given time  $t$ , denoted as  $u(t)$ , is influenced by its previous potential  $u(t_{i-1})$  at time  $t_{i-1}$  and the general pre-synaptic input  $\hat{I}(t)$ . The equation governing this process is given by:

$$u(t) = u(t_{i-1})e^{\frac{t_{i-1}-t}{\tau}} + \hat{I}(t) \quad (1)$$

In this formulation, the term  $e^{\frac{t_{i-1}-t}{\tau}}$  represents the exponential decay of the previous potential over time, where  $e$  acts as the exponential decay factor, and  $\tau$  is the time constant that determines the rate of decay. The second term,  $\hat{I}(t)$ , accounts for the effect of synaptic input received at the current time step, which is the weighted sum of input spikes. This equation describes how the neuron’s potential evolves over time based on both its past state and incoming synaptic activity.

2) *Spiking and reset*: A neuron generates a spike when its membrane potential  $u(t)$  reaches or exceeds a predefined threshold  $\theta$ . Mathematically, this condition is expressed as:

$$u(t) \geq \theta \quad (2)$$

where  $u(t)$  represents the neuron’s membrane potential, and  $\theta$  is the spiking threshold that determines when an action potential is triggered. Our threshold in the SNN is set to one.

After a spike occurs, the membrane potential is reset to a lower value, denoted as  $u_{\text{reset}}$ , to ensure proper neuronal dynamics. This reset mechanism is described by:

$$u(t) \leftarrow u_{\text{reset}} \quad (3)$$

where  $u_{\text{reset}}$  is the reset potential that the neuron returns to immediately after firing a spike. We set the reset potential to zero in our SNN. This mechanism prevents continuous firing and allows the neuron to undergo a refractory period before it can spike again.

### C. Adaptive Pruning for Spiking Neural Network

We propose the adaptive pruning algorithm to prune the synaptic connections of SNNs. The general steps of the algorithm are presented in Figure 2. Compared to regular pruning methods, our approach adapts the pruning rate, fine-tuning interactions, and pruned model based on the ongoing pruning performance determined by the validation dataset. This adaptive behavior reduces the number of training iterations during pruning, and improves the sparsity of the model while maintaining the performance.

We begin the pruning process with a dense SNN pre-trained on the training dataset. The validation loss of this pre-trained network, denoted by  $L_t$ , is used as the target

loss throughout the pruning process. Every pruning iteration prunes the  $p\%$  lowest magnitude weights and is followed by one or more fine-tuning epochs using the training dataset. Initially, the **Starting Pruning Rate** is set at its maximum value, as the network is more robust to pruning when it contains many synapses. After the first iteration, pruning is performed only after the pruned SNN has been fine-tuned to achieve the target loss. Recognizing that the pruned network may not exactly match  $L_t$ , we introduce a **Pruning Tolerance** range around  $L_t$  within which the loss is considered acceptable for further pruning. Additionally, the number of fine-tuning epochs is limited by the **Pruning Patience** parameter. If the network fails to achieve the target loss within the granted epochs, the current pruning rate is deemed too aggressive. In such cases, the most recently pruned model is discarded and the pruning rate is halved. A detailed description of the adaptive pruning algorithm is provided in Algorithm 1. Three major hyperparameters are introduced to control the adaptiveness of our algorithm:

- 1) **Starting Pruning Rate:** Pruning rate for the first pruning iteration and representing the highest pruning rate during the pruning process. The rate influences the pruning speed by directly influencing the pruning magnitude at each step.
- 2) **Pruning Tolerance:** The rate of the pre-trained validation loss that the validation loss of pruned networks is "tolerated" to exceed for executing pruning. This is introduced to take care of random fluctuations of loss during training.
- 3) **Pruning Patience:** The number of fine-tuning epochs to wait before halving the pruning rate and discarding the most recent pruned network.

Our synaptic pruning algorithm can be performed either globally or on a per-layer basis on the SNN. In the global approach, the  $p\%$  lowest-magnitude weights across all hidden layers of the SNN are pruned, whereas in the per-layer approach, the  $p\%$  lowest-magnitude weights in each hidden layer are pruned. We adopt the per-layer pruning approach, as it yields a more balanced pruned SNN compared to the global method. During GPU-based training, pruning is implemented

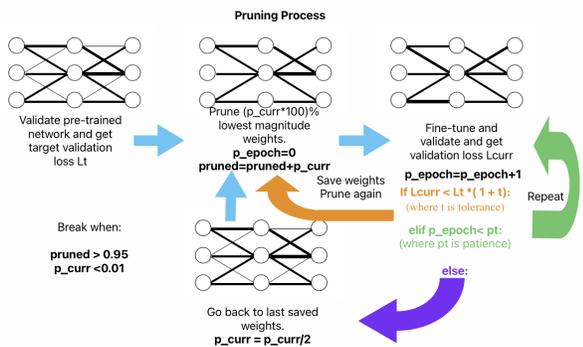


Fig. 2. General steps of our proposed adaptive pruning algorithm for SNN. The pruning process keeps iterating until either the target pruned connection percentage (pruned) is reached or the adaptive pruning rate reaches the minimal value (pr).

---

### Algorithm 1 SNN Adaptive Pruning Algorithm

---

- 1: Minimum pruning rate  $p_{min} \leftarrow 0.1$  and maximum pruning possible  $pruned_{max} \leftarrow 0.95$  are fixed.
  - 2: **Initialize:**
  - 3: Starting Pruning Rate:  $p_{start}$
  - 4: Pruning Patience:  $pt$
  - 5: Pruning Tolerance:  $t$
  - 6: % of network that has been pruned:  $pruned$
  - 7: Validate the network to obtain the initial target loss  $L_t$
  - 8: Set the current pruning rate:  $p_{curr} = p_{start}$
  - 9: **while**  $p_{curr} \geq p_{min}$  **and**  $pruned < pruned_{max}$  **do**
  - 10: Apply pruning with rate  $p_{curr}$
  - 11: Update:  $pruned = pruned + p_{curr}$
  - 12: Initialize current loss:  $L_{curr} \leftarrow \infty$
  - 13: Reset fine-tuning epoch counter:  $p_{epoch} \leftarrow 0$
  - 14: **while**  $L_{curr} > L_t(1 + t)$  **do**
  - 15: **if**  $p_{epoch} > pt$  **then**
  - 16: Restore previously pruned SNN
  - 17: Reduce pruning rate:  $p_{curr} \leftarrow p_{curr}/2$
  - 18: **break**
  - 19: **end if**
  - 20: Fine-tune epoch of the pruned SNN
  - 21: Compute validation loss:  $L_{curr}$
  - 22:  $p_{epoch} \leftarrow p_{epoch} + 1$
  - 23: **end while**
  - 24: **end while**
- 

via element-wise multiplication of the weight matrix with a binary mask, where pruned weights are assigned a value of 0 and retained weights a value of 1. We adopt a static pruning approach in which the same mask is reapplied at every batch during fine-tuning and prior to validation. This ensures that pruned weights do not regain relevance over time.

## III. EXPERIMENTS AND RESULTS

### A. Datasets and Experiments

We benchmarked the performance of our pruned SNN using the Primate Reaching task from the neuromorphic benchmark NeuroBench [15]. The task comprises data from 6 recording sessions obtained from 2 Rhesus primates (Indy and Loco), with each subject contributing 3 sessions [25]. According to the NeuroBench benchmark, each session is subdivided into 4 sub-sessions, within each sub-session 50% of the data is allocated for training, 25% for validation, and 25% for testing. The task output is a two-dimensional prediction of finger velocity, represented by the X and Y coordinates.

Since the NeuroBench benchmark uses a different split for the training, validation, and test datasets than the evaluation presented in [14], we retrained the dense SNN3 model using the NeuroBench split so that it could serve as the pre-trained network for pruning. We applied adaptive pruning to the pre-trained models from all 6 sessions using identical hyperparameters for *Starting Pruning Rate* = 10 and *Pruning Patience* = 5, and different *Pruning Tolerance* = 0.1/0.05 based

on the model validation performance. Furthermore, we conducted experiments to compare various pruning approaches and performed ablation studies using data from the first recording session in the dataset.

### B. NeuroBench Harness and Selected Metrics

To fairly compare with other dense networks, we tested all our pruned SNNs using the NeuroBench Harness library. We selected the following metrics from NeuroBench that are relevant to our experiments:

- 1)  **$R^2$  (coefficient of determination)**: This metric is calculated separately for the  $X$ - and  $Y$ -velocity components and taking the average. To determine the correctness score for a session, the  $R^2$  values for both velocity components are averaged.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (4)$$

where  $n$  is the number of labeled points in the test dataset,  $y_i$  is the groundtruth velocity at each point, the predicted velocity as  $\hat{y}_i$  the predicted velocity and  $\bar{y}$  the mean of groundtruth velocities.

- 2) **Connection Sparsity**: Connection sparsity represents the percentage of zero-value synaptic weights in a neural network. It is computed by dividing the number of zero-value synaptic weights by the overall number of weights in the network.
- 3) **Activation Sparsity**: The average sparsity of neuron activations across all layers, timesteps, and input samples during network execution. A value of 0 indicates that all neurons generate non-zero activation values, while 1 means all neurons generate activation values equal to zero.
- 4) **Effective Synaptic Operations**: The average number of effective synaptic operations during network execution, determined by non-zero neural activations and synaptic weights. This metric is categorized into Multiply-and-Accumulate (MAC) operations for ANNs, and Accumulate (AC) operations for SNNs.

Detailed descriptions of each selected metric can be found in the NeuroBench Harness [15].

### C. Comparisons with Baseline Dense Networks

We evaluated the test performance of our adaptive pruning algorithm relative to dense networks using the selected Neurobench metrics and the benchmarking harness. Table I presents the average results across different recording sessions. Compared to dense SNN and ANN approaches, our sparsely pruned SNN achieves comparable performance while using only 10% of the effective synaptic operations in most sessions. With similar activation sparsity, this drastic reduction in synaptic operations is entirely attributable to the increased connection sparsity introduced by our pruning algorithm. Since the network operation cost on hardware is proportional to the number of synaptic operations, this

reduction directly enhances the energy efficiency of network inference.

Although our pruned network achieves performance comparable to that of dense networks, pruning still results in a reduction in  $R^2$ . To investigate the cause of this performance reduction, Table II presents a detailed comparison of the performance between the pre-trained dense SNN and the pruned SNN for each recording session. We observed that the reduction in  $R^2$  varies across recording sessions. Specifically,  $R^2$  remains relatively stable for *indy\_20160622\_01*, *indy\_20170131\_02*, and *loco\_20170301\_05*, whereas other sessions exhibit more pronounced declines after pruning. This suggests that the pruning process may be overly aggressive for these sessions, likely due to their already extremely high activation sparsity (exceeding 99%), which increases sensitivity to synaptic pruning. To mitigate this effect, we reduced the pruning tolerance for the unstable sessions.

### D. Comparing Global and Per-layer Pruning

Table III compares the single session performance of global and per-layer pruning approaches. Our results indicate that while the global pruning method achieves performance similar to that of the per-layer approach, it exhibits a slightly lower  $R^2$ . The per-layer pruning approach is preferred because it enforces a uniform pruning rate across all layers, resulting in similar connection sparsity within each layer. This uniformity benefits network deployment on neuromorphic hardware, as it ensures that each layer consumes similar computational resources and operates within comparable time constraints.

Figures 3 and 4 show the detailed adaptive pruning process for both per-layer and global approaches on a single recording session. In the figures, the red dotted lines indicate the epochs at which pruning operations are executed, with the network pruned rate after pruning annotated above each line. The shaded lines represent pruning operations that were ultimately discarded because the network failed to reach the target loss after fine-tuning. The pruning process terminates once the exit criteria are met. In both approaches, the process terminates upon reaching the minimum pruning rate. This minimum is achieved after multiple adaptive reductions in the pruning rate, prompted by the network’s failure to achieve the target loss after fine-tuning. Consequently, the total number of fine-tuning epochs exceeds the optimal effective epochs required to obtain the final pruned SNN. However, we argue that these additional training epochs are not wasted, as they represent an exploration process that facilitates further improvements in sparsity.

### E. Ablation Studies on Adaptive Pruning

We conducted two ablation studies to evaluate the contributions of individual components in our adaptive pruning algorithm. Specifically, we investigated the roles of the tolerance and rate decay components. The tolerance component governs when the network initiates pruning based on the validation loss, while the rate decay component discards ineffectively pruned models and reduces the pruning rate

TABLE I  
PERFORMANCE COMPARISON BETWEEN BASELINE DENSE NETWORKS AND OUR PRUNED NETWORKS.

| Method                           | Average of Indy sessions |                     |                     |                      | Average of Loco sessions |                     |                     |                      |
|----------------------------------|--------------------------|---------------------|---------------------|----------------------|--------------------------|---------------------|---------------------|----------------------|
|                                  | $R^2$                    | Connection Sparsity | Activation Sparsity | Effective Operations | $R^2$                    | Connection Sparsity | Activation Sparsity | Effective Operations |
| NeuroBench ANN [15]              | 0.593                    | 0.0                 | 0.683               | 3836 MACs            | 0.558                    | 0.0                 | 0.668               | 6103 MACs            |
| NeuroBench SNN [15]              | 0.593                    | 0.0                 | 0.997               | 276 ACs              | 0.568                    | 0.0                 | 0.999               | 551 ACs              |
| Dense SNN3 [14]                  | 0.583                    | 0.0                 | 0.9813              | 408.14 ACs           | 0.570                    | 0.0                 | 0.9893              | 628.10 ACs           |
| <b>Our adaptive pruning+SNN3</b> | 0.570                    | <b>0.8915</b>       | 0.9826              | <b>38.37 ACs</b>     | 0.546                    | <b>0.791</b>        | 0.9898              | <b>95.38 ACs</b>     |

TABLE II  
PERFORMANCE COMPARISON OF EACH RECORDING SESSION BETWEEN DENSE NETWORKS AND OUR PRUNED NETWORKS.

| Session name     | Method                  | Tolerance | $R^2$  | Connection Sparsity | Activation Sparsity | Effective Operations (ACs) | Pruned Rate (%) |
|------------------|-------------------------|-----------|--------|---------------------|---------------------|----------------------------|-----------------|
| indy_20160622_01 | Dense SNN3[14]          | -         | 0.6618 | 0.0                 | 0.9789              | 535.2                      | -               |
|                  | <b>Adaptive Pruning</b> | 0.1       | 0.6539 | 0.8978              | 0.9763              | 54.63                      | 90              |
| indy_20170131_02 | Dense SNN3[14]          | -         | 0.5812 | 0.0                 | 0.9711              | 391.0                      | -               |
|                  | <b>Adaptive Pruning</b> | 0.1       | 0.5653 | 0.9072              | 0.9786              | 27.88                      | 89              |
| indy_20160630_01 | Dense SNN3[14]          | -         | 0.5065 | 0.0                 | 0.9937              | 298.1                      | -               |
|                  | <b>Adaptive Pruning</b> | 0.1       | 0.4559 | 0.7966              | 0.9930              | 45.67                      | 80              |
|                  | <b>Adaptive Pruning</b> | 0.05      | 0.4919 | 0.8696              | 0.9929              | 32.60                      | 87              |
|                  | Dense SNN3[14]          | -         | 0.5978 | 0.0                 | 0.9852              | 720.3                      | -               |
| loco_20170301_05 | <b>Adaptive Pruning</b> | 0.1       | 0.5805 | 0.9062              | 0.9890              | 50.91                      | 91              |
|                  | Dense SNN3[14]          | -         | 0.5475 | 0.0                 | 0.9919              | 608.0                      | -               |
| loco_20170215_02 | <b>Adaptive Pruning</b> | 0.1       | 0.4859 | 0.8810              | 0.9883              | 48.57                      | 88              |
|                  | <b>Adaptive Pruning</b> | 0.05      | 0.5145 | 0.6678              | 0.9911              | 155.85                     | 66              |
| loco_20170210_03 | Dense SNN3[14]          | -         | 0.5648 | 0.0                 | 0.9906              | 555.8                      | -               |
|                  | <b>Adaptive Pruning</b> | 0.1       | 0.5295 | 0.8714              | 0.9884              | 45.60                      | 87              |
|                  | <b>Adaptive Pruning</b> | 0.05      | 0.5424 | 0.7991              | 0.9893              | 79.37                      | 79              |

TABLE III  
COMPARISON OF GLOBAL AND LAYER-WISE PRUNING APPROACHES.

| Metric              | Global Pruning | Layer Pruning |
|---------------------|----------------|---------------|
| $R^2$               | 0.6333         | 0.6539        |
| Connection Sparsity | 0.8811         | 0.8978        |
| Activation Sparsity | 0.9724         | 0.9763        |
| Effective ACs       | 57.61          | 54.63         |
| Pruned Rate (%)     | 89%            | 90%           |
| Total Epoch         | 45             | 39            |
| Optimal Epoch       | 33             | 21            |

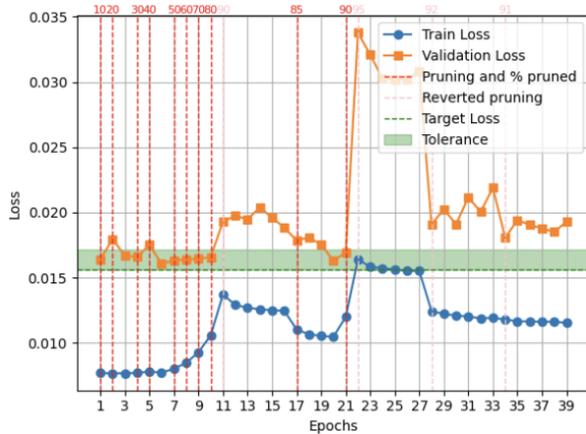


Fig. 3. Training and validation loss during the adaptive pruning process using the **per-layer** approach.



Fig. 4. Training and validation loss during the adaptive pruning process using the **global** approach.

over time. In the first study, we implement fixed pruning by omitting both the tolerance and rate decay components. In this scheme, 10% of the network is pruned after every 5 epochs of fine-tuning, and the process continues until the overall pruning rate reaches 90%. In the second study, we incorporate the tolerance component, enabling the pruning process to decide when to prune based on the validation loss. However, this study omits rate decay. The pruning process terminates when the target loss cannot be achieved within the

TABLE IV  
PERFORMANCE COMPARISON OF DIFFERENT ABLATION STUDIES.

| Ablations |            | $R^2$ | Connection Sparsity | Activation Sparsity | Effective Operations (ACs) | Pruned Rate (%) | Total Epoch | Optimal Epoch |
|-----------|------------|-------|---------------------|---------------------|----------------------------|-----------------|-------------|---------------|
| Tolerance | Rate Decay |       |                     |                     |                            |                 |             |               |
| No        | No         | 0.515 | 0.902               | 0.975               | 55.76                      | 90%             | 40          | -             |
| Yes       | No         | 0.661 | 0.812               | 0.977               | 102.88                     | 80%             | 16          | 10            |
| Yes       | Yes        | 0.654 | 0.898               | 0.976               | 54.63                      | 90%             | 33          | 21            |

number of fine-tuning epochs specified by **Pruning Patience**.

Table IV presents the performance comparison between the ablation studies and the full adaptive pruning algorithm on a single recording session. Compared to the fixed pruning approach, our adaptive pruning method achieves higher  $R^2$  while maintaining the same connection sparsity and requiring fewer training epochs. These results demonstrate the effectiveness of the adaptive approach in achieving rapid pruning and high performance. Furthermore, compared to a variant that employs only the tolerance component, the full adaptive algorithm achieves higher connection sparsity and reduces the effective number of synaptic operations by nearly 50%, while maintaining similar  $R^2$ . These findings indicate that incorporating rate decay and adaptive rollback on aggressively pruned networks enables the pruning process to achieve higher sparsity without sacrificing performance.

#### F. Hardware Simulation on Neuromorphic Processor

To evaluate the hardware benefits of our adaptive pruning algorithm, we conducted a hardware simulation study using realistic measurements obtained from the SENECA neuromorphic processor [26]. SENECA is an energy-efficient, flexible digital neuromorphic processor designed for sparse, event-driven neural networks [23]. To implement our sparse SNN on SENECA, we programmed two micro-kernels to handle spike integration and membrane potential updates for LIF neurons. On SENECA, integrating a single spike (1 AC) into a LIF neuron incurs an energy cost of 12.7 pJ, while updating a LIF neuron at each timestep requires 14.6 pJ. Table V presents the hardware simulation results for one timestep of both the dense and pruned SNNs based on benchmarking results from one recording session. The SNN inference timestep corresponds to 4 ms of neural decoding, and this time bin is used to compute the average power consumption of the SNN. The results demonstrate that our adaptive pruning approach significantly reduces the energy cost of neural decoding and requires minimal power that can potentially support intracortical decoding.

TABLE V  
NEUROMORPHIC HARDWARE SIMULATION COMPARISON.

| Method     | Energy Cost (pJ) | Power ( $\mu$ W) |
|------------|------------------|------------------|
| Dense SNN  | 6811.6           | 1.7              |
| Pruned SNN | 708.4            | 0.18             |

#### IV. CONCLUSION AND DISCUSSION

In this work, we introduce an adaptive pruning algorithm designed to reduce synaptic operations in SNNs for intracortical neural decoding. Our experimental findings indicate that this strategy effectively increases connection sparsity, thereby lowering computational demands and enhancing energy efficiency in brain-machine interfaces. By selectively eliminating redundant synaptic connections, our approach optimizes performance without compromising decoding accuracy, paving the way for more energy-efficient neural implant solutions.

Our adaptive pruning algorithm is specifically designed for SNNs that exhibit high activation sparsity, a property that makes them particularly harmed by performance degradation during pruning. While our approach has been demonstrated on a baseline SNN architecture, its modular design facilitates extension to more complex recurrent SNN models for neural decoding [27]. Adapting the algorithm for recurrent networks will require a balancing strategy that distributes pruning between recurrent and feedforward connections. Given that SNNs can effectively perform spatiotemporal processing even in the absence of recurrent connections, it is possible that prioritizing the pruning of recurrent connections could be achieved without compromising overall performance.

Our adaptive pruning algorithm can be synergistically integrated with complementary network compression techniques, such as quantization [28], to further enhance the efficiency of intracortical neural decoding. Applying quantization to the remaining synaptic weights not only reduces memory requirements but also lowers the energy cost per synaptic operation. Dedicated neuromorphic hardware is essential to fully leverage the efficiency gains of the pruned SNN [29], [30]. However, current neuromorphic platforms may not adequately address the unique requirements of intracortical neural implants [31]. Hence, further hardware development is necessary to realize the full benefits of our approach in practical applications.

#### REFERENCES

- [1] S. Buccelli, Y. Bornat, I. Colombi, M. Ambroise, L. Martines, V. Pasquale, M. Bisio, J. Tessadori, P. Nowak, F. Grassia, A. Aversa, M. Tedesco, P. Bonifazi, F. Difato, P. Massobrio, T. Levi, and M. Chiappalone, "A neuromorphic prosthesis to restore communication in neuronal networks," *iScience*, vol. 19, pp. 402–414, 2019.
- [2] M. A. Cervera, S. R. Soekadar, J. Ushiba, J. Del R Millán, M. Liu, N. Birbaumer, and G. Garipelli, "Brain-computer interfaces for post-stroke motor rehabilitation: a meta-analysis," *Wiley Online Library*, vol. 5, pp. 651–663, 2018.

- [3] C. Lee, B. Kim, J. Kim, S. Lee, T. Jeon, W. Choi, S. Yang, J.-H. Ahn, J. Bae, and Y. Chae, "A miniaturized wireless neural implant with body-coupled power delivery and data transmission," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 11, pp. 3212–3227, 2022.
- [4] S. Miziev, W. A. Pawlak, and N. Howard, "Comparative analysis of energy transfer mechanisms for neural implants," *Frontiers in Neuroscience*, vol. 17, p. 1320441, 2024.
- [5] F. R. Willett, D. T. Avansino, L. R. Hochberg, J. M. Henderson, and K. V. Shenoy, "High-performance brain-to-text communication via handwriting," *Nature*, vol. 593, no. 7858, pp. 249–254, 2021.
- [6] C. Serrano-Amenos, F. Hu, P. T. Wang, S. Kellis, R. A. Andersen, C. Y. Liu, P. Heydari, A. H. Do, and Z. Nenadic, "Thermal analysis of a skull implant in brain-computer interfaces," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2020, pp. 3066–3069.
- [7] F. Liu, S. Meamardoost, R. Gunawan, T. Komiyama, C. Mewes, Y. Zhang, E. Hwang, and L. Wang, "Deep learning for neural decoding in motor cortex," *Journal of Neural Engineering*, vol. 19, no. 5, p. 056021, 2022.
- [8] M. W. Mathis, A. P. Rotondo, E. F. Chang, A. S. Tolias, and A. Mathis, "Decoding the brain: From neural representations to mechanistic models," *Cell*, vol. 187, no. 21, pp. 5814–5832, 2024.
- [9] B. Yin, F. Corradi, and S. M. Bohté, "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks," *Nature Machine Intelligence*, vol. 3, no. 10, pp. 905–913, 2021.
- [10] F. Paredes-Vallés, J. Hagenaaars, J. Dupeyroux, S. Stroobants, Y. Xu, and G. de Croon, "Fully neuromorphic vision and control for autonomous drone flight," *Science Robotics*, vol. 9, no. 90, p. eadi0591, 2024.
- [11] N. Kumar, G. Tang, R. Yoo, and K. P. Michmizos, "Decoding eeg with spiking neural networks on neuromorphic hardware," *Transactions on Machine Learning Research*, 2022.
- [12] Y. Xu, G. Tang, A. Yousefzadeh, G. C. de Croon, and M. Sifalakis, "Event-based optical flow on neuromorphic processor: Ann vs. snn comparison based on activation sparsification," *Neural Networks*, p. 107447, 2025.
- [13] A. Tavanaci, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural networks*, vol. 111, pp. 47–63, 2019.
- [14] P. Hueber, G. Tang, M. Sifalakis, H.-P. Liaw, A. Micheli, N. Tomen, and Y.-H. Liu, "Benchmarking of hardware-efficient real-time neural decoding in brain-computer interfaces," *Neuromorphic Computing and Engineering*, vol. 4, no. 2, p. 024008, 2024.
- [15] J. Yik, K. Van den Berghe, D. den Blanken, Y. Bouhadjar, M. Fabre, P. Hueber, W. Ke, M. A. Khoei, D. Kleyko, N. Pacik-Nelson *et al.*, "The neurobench framework for benchmarking neuromorphic computing algorithms and systems," *Nature Communications*, vol. 16, no. 1, p. 1545, 2025.
- [16] B. Chatterjee, M. Nath, G. Kumar K, S. Xiao, K. Jayant, and S. Sen, "Biphasic quasistatic brain communication for energy-efficient wireless neural implants," *Nature Electronics*, vol. 6, no. 9, pp. 703–716, 2023.
- [17] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [18] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.
- [19] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299.
- [20] Y. He and L. Xiao, "Structured pruning for deep convolutional neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2023.
- [21] M. Gupta, E. Camci, V. R. Keneta, A. Vaidyanathan, R. Kanodia, C. Foo, M. Wu, and J. Lin, "Is complexity required for neural network pruning? a case study on global magnitude pruning," *arXiv preprint*, January 2024.
- [22] Y. Li, Q. Xu, J. Shen, H. Xu, L. Chen, and G. Pan, "Towards efficient deep spiking neural networks construction with spiking activity based pruning," *arXiv preprint*, June 2022.
- [23] Y. Xu, K. Shidqi, G.-J. van Schaik, R. Bilgic, A. Dobrita, S. Wang, R. Meijer, P. Nembhani, C. Arjmand, P. Martinello *et al.*, "Optimizing event-based neural networks on digital neuromorphic architecture: a comprehensive design space exploration," *Frontiers in Neuroscience*, vol. 18, p. 1335422, 2024.
- [24] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal back-propagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018.
- [25] J. G. Makin, J. E. O'Doherty, M. M. Cardoso, and P. N. Sabes, "Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm," *Journal of neural engineering*, vol. 15, no. 2, p. 026010, 2018.
- [26] G. Tang, A. Safa, K. Shidqi, P. Detterer, S. Traferro, M. Konijnenburg, M. Sifalakis, G.-J. van Schaik, and A. Yousefzadeh, "Open the box of digital neuromorphic processor: Towards effective algorithm-hardware co-design," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.
- [27] T. Liu, J. Gyax, J. Rossbroich, Y. Chua, S. Zhang, and F. Zenke, "Decoding finger velocity from cortical spike trains with recurrent spiking neural networks," in *2024 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2024, pp. 1–5.
- [28] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.
- [29] Y. Chen, W. Ye, Y. Liu, and H. Zhou, "Sibrain: A sparse spatio-temporal parallel neuromorphic architecture for accelerating spiking convolution neural networks with low latency," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.
- [30] Y. Kuang, X. Cui, Z. Wang, C. Zou, Y. Zhong, K. Liu, Z. Dai, D. Yu, Y. Wang, and R. Huang, "Essa: Design of a programmable efficient sparse spiking neural network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1631–1641, 2022.
- [31] Y. Wang, X. Yang, X. Zhang, Y. Wang, and W. Pei, "Implantable intracortical microelectrodes: reviewing the present with a focus on the future," *Microsystems & Nanoengineering*, vol. 9, no. 1, p. 7, 2023.