A PyTorch-Compatible Spike Encoding Framework for Energy-Efficient Neuromorphic Applications

Alexandru Vasilache[1,2], Jona Scholz[1], Vincent Schilling[2], Sven Nitzsche[1,2],
Florian Kaelber[3], Johannes Korsch[3], Juergen Becker[2]
[1] *FZI Research Center for Information Technology, Karlsruhe, Germany*
[2] *Karlsruhe Institute of Technology, Karlsruhe, Germany*
[3] *NXP Semiconductors Germany GmbH, Munich, Germany*
email:{vasilache,jona.scholz,nitzsche}@fzi.de
utmlb@student.kit.edu Juergen.Becker@kit.edu
{florian.kaelber,johannes.korsch}@nxp.com

***Abstract*—Spiking Neural Networks (SNNs) offer promising energy efficiency advantages, particularly when processing sparse spike trains. However, their incompatibility with traditional datasets, which consist of batches of input vectors rather than spike trains, necessitates the development of efficient encoding methods. This paper introduces a novel, open-source PyTorch-compatible Python framework for spike encoding, designed for neuromorphic applications in machine learning and reinforcement learning. The framework supports a range of encoding algorithms, including Leaky Integrate-and-Fire (LIF), Step Forward (SF), Pulse Width Modulation (PWM), and Ben's Spiker Algorithm (BSA), as well as specialized encoding strategies covering population coding and reinforcement learning scenarios. Furthermore, we investigate the performance trade-offs of each method on embedded hardware using C/C++ implementations, considering energy consumption, computation time, spike sparsity, and reconstruction accuracy. Our findings indicate that SF typically achieves the lowest reconstruction error and offers the highest energy efficiency and fastest encoding speed, achieving the second-best spike sparsity. At the same time, other methods demonstrate particular strengths depending on the signal characteristics. This framework and the accompanying empirical analysis provide valuable resources for selecting optimal encoding strategies for energy-efficient SNN applications.**

***Keywords*-Spiking Neural Networks (SNNs); spike encoding; neuromorphic; energy-efficiency; Reinforcement Learning (RL).**

I. INTRODUCTION

An increasing number of tasks rely on Machine Learning (ML) to enhance accuracy, streamline development processes, or facilitate automation in the first instance. Neural Networks (NNs) are often employed for this purpose due to their flexibility and capacity to solve even the most complex tasks. As neural networks become more widespread, they are also being employed with increasing frequency in embedded systems. However, their deployment in such embedded contexts is frequently constrained by the requisite computing power and the associated high energy demands. Given these considerations, SNNs may offer a promising avenue for integrating high-performance machine learning into embedded systems, provided that suitable neuromorphic hardware is available.

In order to utilize SNNs, it is necessary to have access to adequate spiking data, also referred to as event-based data. Moreover, the event-based data must be sparse to leverage the advantages of spiking neural networks fully. Ideally, an event-based sensor would be employed to provide this data. However, there is a notable scarcity of such sensors on the market. Currently, the only commercially available event-based sensors target the vision modality through event-based cameras [1][2].

As an alternative approach, non-spiking data can be converted using spike encoding algorithms. Such algorithms are designed to transform real-valued data into spike trains, which can then be utilized as input for an SNN. The existing literature classifies spike encoding algorithms into two principal categories: rate coding and temporal coding [3]. In rate coding, the signal amplitudes are directly mapped to spike frequencies, resulting in high spiking activity. In contrast, temporal coding results in the firing of spikes only when specific events occur, thereby making the timing of spikes crucial and reducing the overall number of spikes. This sparsity renders temporal coding more power-efficient and suitable for low-power applications. This paper focuses on evaluating four temporal encoding methods: Leaky Integrate-and-Fire (LIF), Step-Forward (SF), Pulse Width Modulation (PWM), and Ben's Spiker Algorithm (BSA). These methods were chosen as they are frequently discussed and compared in spike encoding literature [4][5][6][7], serving as representative techniques for this study.

Even though many spike encoding methods are publicly available, no open-source framework or library currently groups various such algorithms and allows for straightforward integration into popular machine-learning workflows. Accordingly, this work seeks to provide a framework that offers out-of-the-box PyTorch support and automatic parameter optimization of the encoding algorithms for a given dataset. Additionally, we evaluate the performance of various spike encoding algorithms for specific signal types and implement them on an embedded platform to assess runtime and power demand. The library is publicly accessible at its GitHub Repository [8]. This work aims to reduce the overhead associated with integrating SNNs into real-world applications.

The remainder of this paper is organized as follows: Section II compares our work with existing state-of-the-art approaches. Section III then summarizes the investigated methods and details the experimental setup, followed by Section IV

which displays the results regarding reconstruction error, spike sparsity, runtime, and power consumption. Subsequently, Section V discusses these results focusing on energy efficiency and accuracy for different signal types. Finally, Section VI outlines the structure of the provided spike encoding library.

II. Related Work

A. Evaluation of Spike Encoding Algorithms

Wang et al. [4] implemented four spike encoding algorithms on a Field Programmable Gate Array (FPGA) and compared them by their speed, power consumption, accuracy, and robustness to noise. They evaluated Sliding Window (SW) encoding [9], BSA [10], SF encoding [11] and the PWM algorithm [12]. Furthermore, they verified their evaluation results on two real-world applications: tactile signal encoding reconstruction and a robotic arm control task. They found that overall BSA had the highest power consumption and was therefore not recommended for most applications, with the notable exception of encoding square signals, where its energy efficiency was high. PWM performed well in most signal reconstruction tests, and its simple implementation made it favorable for real-world use. However, its accuracy may be decreased if an unsuitable curve-fitting algorithm is chosen. Also, if nonlinear operations are used, the power consumption may be elevated. This added variability is reduced in SF encoding, which only has one adjustable parameter. While SF encoding also performed well in most signal reconstruction tests, its accuracy could degrade when high changes in the signal amplitude occurred. Finally, SW encoding had no notable advantages over the other algorithms.

Chen et al. [5] compared (among others) SF to PWM and provided further evidence that both algorithms achieve high signal reconstruction accuracy. They observed that SF is more accurate at lower thresholds, but the increased spike count may also increase power consumption.

In [6], the authors suggest a workflow for selecting, optimizing, and validating spike encoding methods, which they evaluate on SF, BSA, and others. Their experiments provide further evidence for SFs versatility and robustness, consistent with similar studies' findings. However, their evaluation of BSA contradicts those of [4] regarding step signals since they observed high signal reconstruction errors for this signal type. An implementation choice may partly explain this since they use a multiplicative threshold rather than a subtractive one.

Yarga et al. [7] applied BSA, LIF encoding, and three additional methods to encode voice recordings. Their results are not fully applicable to our comparison, as they focused on the accuracy of a classification task rather than evaluating reconstruction error. Initially, features were extracted into a spectrogram or cochleagram and then converted to spike trains using the evaluated encoding methods. These spike trains were subsequently processed by a Convolutional Neural Network (CNN) for classification. Their evaluation metrics included spike density and classification accuracy, which are indirectly related to the energy efficiency of the encoding and its impact on information loss or gain. However, these metrics may not fully represent the performance of the encoding methods. Most methods allowed spike density to be adjusted by modifying their parameters, such as the membrane threshold in LIF encoding. On spectrogram features, their findings showed that LIF encoding produced the highest accuracy across most spike densities among the evaluated methods. Notably, when using spectrogram features, BSA also performed well at very low spike densities. However, on cochleagram features, BSA performed poorly, and LIF was outperformed by other methods at low spike densities. At higher spike densities, LIF once again achieved the highest overall classification accuracy. In summary, their findings suggest that LIF encoding can achieve both high classification accuracy and low spike densities, contributing to better energy efficiency. In contrast, BSA is more restricted to specific scenarios where it performs well.

B. Spike Encoding Repositories

The currently available spike encoding repositories tend to prioritize applications outside the domain of machine learning. SpikeCoding [13] emphasizes the real-time control of robots and has been developed to integrate with ROS rather than focusing on machine learning workflows. Similarly, Spikes [14] offers foundational tools for spike generation but is not designed for integration with neural network models. In contrast, the proposed framework addresses this limitation by providing a PyTorch-compatible, open-source library focused on machine learning and neural network applications, including support for reinforcement learning environments, facilitating broader use in machine learning research and practice.

III. Methods

This section presents the investigated encoding methods and their evaluation on an embedded hardware platform and outlines the experimental setup. This includes an explanation of the underlying hardware processes, the types of signals used, and the parameter optimization of the encoding algorithms.

A. Encoding Algorithms

SF encoding is a straightforward and efficient method for signal processing. A spike is generated whenever the signal surpasses a defined baseline, which is subsequently incremented by a constant value. Conversely, if the signal did not exceed the baseline, the baseline is lowered by a constant value. A pseudocode implementation is shown in Figure 1 (based on [15]). Encoding spikes in this way is similar to the LIF encoding method, although the latter is inspired by a spiking neuron model of the same name.

In LIF encoding, the signal serves as an input current that increases the membrane potential. When the potential exceeds a predefined threshold, a spike is emitted. The membrane potential decays proportionately to a decay variable at each time step. It is important to note that the original signal must be normalized, as neither the decay rate nor the threshold adapts to the varying range of possible signal values. A pseudocode implementation of the LIF encoding method is provided in

```
Input: signal, threshold
Output: spike_train
base ← 0, up_spikes ← 0, down_spikes ← 0
for t = 1 to time_steps do
    if signal(t) > base + threshold then
        up_spikes(t) ← 1
        base ← base + threshold
    else if signal(t) < base − threshold then
        down_spikes(t) ← −1
        base ← base − threshold
    else
        up_spikes(t) ← 0
        down_spikes(t) ← 0
    end if
end for
spikes ← up_spikes + down_spikes
```

Figure 1. SF Encoding Method

```
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
```

Figure 2. LIF Encoding Method

Figure 2 (based on [7]), where min_max_normalize refers to a rescaling of the range of features to $[0, 1]$.

Similar to LIF encoding, BSA requires data normalization but employs a fundamentally different approach. Instead of encoding a continuous signal into spikes, BSA assumes the signal has already been transformed into a spike train. The objective is to decode this spike train with minimal reconstruction error. The encoding process is assumed to involve convolution with a Finite Impulse Response (FIR) filter, and decoding requires reversing this operation through deconvolution. At each step, an error term is computed to measure the sum of the differences between the filter and the signal, while a second error term quantifies the sum of the signal itself. Spikes are emitted when the first error term is smaller than the second minus a threshold value. Upon spike emission, the filter is subtracted from the signal. A pseudocode implementation of BSA encoding is provided in Figure 3 (based on [16]).

Finally, PWM is based on comparing the input signal to a carrier (or "reference") signal and emitting spikes whenever the carrier signal exceeds the input signal. Once this condition is met, the input signal must first fall below the carrier signal before exceeding it again for a new spike to be emitted. The carrier signal is a sawtooth wave with an adjustable frequency

```
Input: signal, filter_order, filter_cutoff, threshold
Output: spike_train
signal ← normalize(signal)
fir_coeff ← fir_filter(filter_size = filter_order + 1, filter_cutoff, sampling_frequency = 1)
spikes ← 0
for t = 1 to time_steps do
    err1 ← 0
    err2 ← 0
    for j = 1 to filter_size do
        if t + j − 1 ≤ time_steps then
            err1 ← err1 + |signal(t + j − 1) − fir_coeff(j)|
            err2 ← err2 + |signal(t + j − 1)|
        end if
    end for
    if error1 ≤ err2 − threshold then
        spikes(t) ← 1
        for j = 1 to filter_size do
            if t + j − 1 ≤ time_steps then
                signal(t + j − 1) ← signal(t + j − 1) − fir_coeff(j)
            end if
        end for
    else
        spikes(t) ← 0
    end if
end for
```

Figure 3. BSA Encoding Method

parameter. As with the other encoding methods, the input signal must be normalized to ensure it overlaps with the carrier signal. A pseudocode implementation of the PWM encoding method is shown in Figure 4 (based on [17]).

```
Input: signal, frequency, downspike (boolean)
Output: spike_train
signal ← normalize(signal)
carrier ← sawtooth(frequency)
neg_carrier ← 1 − carrier
pwm ← 0, up_spikes ← 0, down_spikes ← 0
for t = 1 to time_steps do
    if signal(t) < carrier(t) then
        pwm(t) ← 1
    else if signal(t) > neg_carrier(t) and downspike = True then
        pwm(t) ← −1
    else
        pwm(t) ← 0
    end if
end for
for t = 2 to time_steps do
    if pwm(t) = 1 and pwm(t − 1) ≠ 1 then
        up_spikes(t) ← 1
    else if pwm(t) = −1 and pwm(t − 1) ≠ −1 then
        down_spikes(t) ← −1
    end if
end for
spikes ← up_spikes + down_spikes
```

Figure 4. PWM Encoding Method

B. Experimental setup

To evaluate the performance characteristics, such as runtime and power consumption, we utilized specific embedded hardware. While the core encoding algorithms are implemented in C++, the subsequent performance results presented are specific

to the chosen platform and configuration. We use an MCX-N947-EVK development board from NXP Semiconductors as an embedded evaluation platform. The MCX N is a multi-core System on Chip comprising two Arm Cortex®-M33 cores, 2MB flash, 512kB SRAM, a digital signal co-processor, and a neural processing unit for accelerating inference of conventional neural networks. One Cortex-M33 has a maximum clock speed of 150MHz and offers a favorable trade-off between energy efficiency, real-time determinism, and system security. The power consumption comes down to 57 µA/MHz in active mode. For our evaluation, we configured the system's clock frequency to 12 MHz and set the power mode to overdrive with a DCDC core voltage of 1.2V, balancing power consumption and performance.

The interaction with the development board is facilitated by eRPC calls, wherein the board is treated as a server and the PC as a client. First, the client transmits its signal data to the board via UART. It then requests the execution of the selected encoding function on the transmitted data. Finally, the client requests the results once encoding is completed, and the board transfers them back. All measurements are performed between the start and end of the encoding method to exclude data transfer and communication overhead. Additionally, the normalization operations required by some methods have been excluded.

The power consumption of each method was measured over 10 hours at a sample rate of 62500 samples per second, with the results averaged over the entire duration. It should be noted that even when no operation is performed, the board continues to consume power. Therefore, a baseline power consumption measurement was first taken and subsequently subtracted from each method's measurements, allowing for a direct comparison between the encoding methods. The time measurements were taken using C functions, which initiated and terminated a timer at the beginning and end of method execution, respectively.

In order to quantify the extent of information loss incurred during the encoding process, we calculate the reconstruction error for each encoding method by decoding the encoded signal and measuring the Mean Squared Error (MSE) between the reconstruction and the original signal.

C. Evaluated Signals

This study selected four artificially generated signals with 16384 time steps, each normalized to unit mean and variance, to assess the encoding methods. These specific signals were chosen to represent a diverse set of common temporal dynamics, including irregularity, drift, abrupt transitions, and smooth periodicity.

The "Vibration" signal (Figure 5 (a)) simulates oscillations with a specific standard deviation and frequent fluctuations, challenging the encoders due to its irregularity. The "Trended" signal (Figure 5 (b)) introduces drift, testing the stability of the reconstruction over time. The "Rectangular" signal (Figure 5 (c)) resembles a square wave, providing insights into the methods' performance on constant and abruptly transitioning

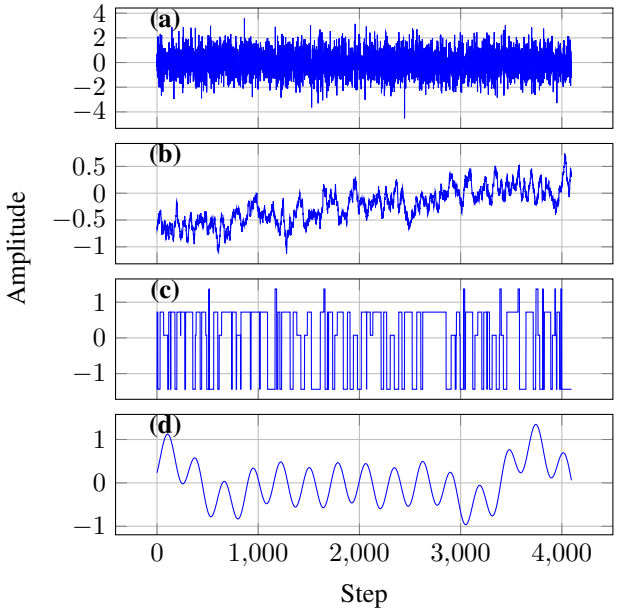

Figure 5. Four different signal types: (a) Vibration Signal, (b) Trended Signal, (c) Rectangular Signal, (d) Sinusoidal Signal.

values. Lastly, the "Sinusoidal" signal (Figure 5 (d)), smooth and noise-free, assesses the preservation of periodic patterns.

D. Paramter Optimization

Each encoding method was optimized through 500 trials. In each trial, the chosen set of parameters was used to encode the signal into spikes. The resulting spike train was then decoded to reconstruct the original signal, and the reconstruction error, quantified as the MSE between the original and reconstructed signals, was computed. The optimization aimed to minimize this reconstruction error by adjusting the encoding parameters.

To perform this optimization, we employed Optuna [18], a widely used hyperparameter optimization framework implemented in Python. For most encoding methods, random sampling was used to explore the parameter space randomly. However, for the BSA method, which involves tuning three interdependent parameters and is computationally demanding, the TPESampler [19] was utilized due to its efficiency in complex, multi-variable optimizations.

IV. Results

This section presents the results for each encoding method applied to different signal types, evaluated using four main criteria: reconstruction error (TABLE I), encoding time, power consumption (TABLE III), and spike sparsity (TABLE II). Spike sparsity is expressed as the ratio of spikes to the total signal length (16384).

The absolute power values reported in TABLE III reflect the average power consumption measured over ten hours for the entire board. Additionally, the absolute power was measured for the no-operation (NOP), resulting in a value of 99.74 nW. The NOP reference power is subtracted from this value to determine the dynamic power consumed by each encoding

TABLE I. RECONSTRUCTION ERROR (MSE)

Encoding Method	LIF	SFC	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

TABLE II. SPIKE SPARSITY

Encoding Method	LIF (%)	SFC (%)	PWM (%)	BSA (%)
Vibration	**36.83**	41.26	50.00	53.81
Trended	65.59	35.82	**3.66**	61.34
Rectangular	62.47	**10.20**	14.33	48.48
Sinusoidal	42.72	35.14	**3.02**	48.70
Mean Sparsity	51.90	30.60	**17.75**	53.08

TABLE III. TIME, POWER AND ENERGY MEASUREMENTS

Encoding Method	LIF	SF	PWM	BSA
time (ms)	127.15	**123.52**	691.32	2238.58
absolute power (nW)	109.45	**106.61**	111.15	119.74
dynamic power (nW)	9.71	**6.87**	11.41	20.00
dynamic energy (pW h)	0.34	**0.24**	2.19	12.44
time wrt. SF (%)	2.94	**0.00**	459.64	1712.28
dynamic power wrt. SF (%)	41.34	**0.00**	66.08	191.12
dynamic energy wrt. SF (%)	41.67	**0.00**	812.50	5083.33

method. The last three rows of the table indicate the encoding time, dynamic power, and dynamic energy consumption for each method, expressed as a percentage increase relative to the SF converter, which shows the lowest values across all methods.

V. Discussion

This study evaluated four encoding methods—LIF, SF, PWM, and BSA—based on their performance in reconstructing signals, spike sparsity, and energy efficiency.

LIF demonstrated high accuracy with vibration data due to its suitability for signals fluctuating around a baseline. However, its mean bias hampered performance with trended signals and capturing extreme values in sinusoidal signals. Despite a high spike rate, its simple implementation resulted in good energy efficiency, aligning with findings from Yarga et al. [7] on its suitability for fluctuating signals.

The SF Converter generally achieved the lowest reconstruction error, especially with sinusoidal signals, and offered superior speed and energy efficiency. However, it exhibited instability with rectangular waveforms. Our results confirm prior evidence from Chen et al. [5] on SF's accuracy, showing it had the lowest mean reconstruction error (0.1614 - TABLE I) and minimal power consumption (TABLE III), underscoring its efficiency and versatility.

PWM, despite low spike sparsity, underperformed in reconstruction accuracy, struggling with fluctuating and rectangular waveforms. Its complex operations resulted in higher encoding time and power consumption. This contrasts with Wang et al. [4], who reported favorable reconstruction and power consumption for PWM in simpler implementations. Our results indicate PWM's higher reconstruction error (mean MSE of 0.3227) and elevated power consumption (812.5% relative to SF's energy usage), likely due to its sensitivity to parameter tuning.

BSA effectively filtered high frequencies and achieved low reconstruction error for rectangular signals (MSE of 0.0636) but suffered from high computational costs and sensitivity to initial signal values, consistent with Wang et al. [4]. This limits its practicality for real-time embedded systems, as its encoding duration and power consumption are significantly higher than SF.

Overall, SF emerged as the most energy-efficient and reliable method, while LIF and BSA provide niche strengths in specific applications. Future work may benefit from investigating hybrid approaches.

```python
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 6. Using the StepForwardConverter to encode a signal.

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

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

Figure 7. Minimal example of optimizing a converter and decoding signals in order to reconstruct them.

VI. Spike Encoding Framework

Our framework is based on the idea of a Converter. In our approach, a Converter is an object that can encode and decode signals. Optionally, Converters may also include a method of optimization that allows them to finetune their hyperparameters to a specific signal. In this way, each encoding method can achieve its highest reconstruction accuracy without the need to finetune its parameters manually.

For example, consider using our SF implementation depicted in Figure 6. LIF, PWM and BSA encoding all function analogously.

Decoding spike trains is achieved similarly, except that *encode* is replaced with *decode*. In Figure 7, we see how converters can be optimized and used to decode signals.

Additionally, we introduce a custom encoder designed explicitly for Gymnasium [20] environments, the GymnasiumEncoder. It extends rate encoding methods by adding utility methods tailored for reinforcement learning tasks. We also provide the BinEncoder, which utilizes Gaussian Receptive Field (GRF) encoding to transform single values into multiple bin responses. However, like the GymnasiumEncoder, it is restricted to encoding operations only.

VII. Conclusion and Future Work

This paper introduces a novel, open-source PyTorch-compatible Python framework for spike encoding, explicitly designed for machine learning and reinforcement learning applications. The framework offers support for a diverse range of encoding methods, encompassing conventional algorithms, such as LIF, SF, PWM, and BSA, as well as specialized components like a reinforcement learning-optimized encoder and Gaussian Receptive Field-based population coding. The framework is accompanied by documentation and testing, ensuring seamless integration into machine learning workflows and fostering accessibility and ease of use.

Furthermore, a comprehensive evaluation of the performance trade-offs of each encoding method was conducted by implementing them in C++ and testing them on embedded hardware. Our findings indicate that while SF exhibited the highest energy efficiency and fastest encoding time, it tended to falter in abrupt signal transitions. LIF demonstrated efficacy in handling fluctuating signals but exhibited limitations in the presence of trends or extreme values. PWM demonstrated lower accuracy and higher energy consumption than the other methods. In contrast, BSA demonstrated high accuracy for certain signal types and filtering capabilities but at the cost of increased computational demands. These comparative insights provide valuable guidance for selecting the most suitable encoding method, supporting the broader adoption of Spiking Neural Networks in machine learning applications.

Future work will focus on extending the framework's capabilities by implementing additional encoding algorithms, including both temporal and rate coding approaches, to allow for broader comparisons.

ACKNOWLEDGMENTS

This research is funded by the German Federal Ministry of Education and Research as part of the project "ThinKIsense", funding no. 16ME0564.

Supplementary Materials

We provide Python implementations for all the investigated algorithms. Our repository includes LIF encoding, SF, PWM, and BSA, as well as two additional algorithms. The first is a custom encoder particularly suited to reinforcement learning, especially to Gymnasium [20] environments. The second one implements a form of population coding that is based on Gaussian Receptive Fields. Our code can be accessed at [8].

References

[1] C. Aerne, *iniVation – Neuromorphic vision systems*, en-US.

[2] *PROPHESEE | Metavision Technologies*, en-US.

[3] D. Auge, J. Hille, E. Mueller, and A. Knoll, "A survey of encoding techniques for signal processing in spiking neural networks", *Neural Processing Letters*, vol. 53, no. 6, pp. 4693–4710, 2021.

[4] K. Wang, X. Hao, J. Wang, and B. Deng, "Comparison and selection of spike encoding algorithms for snn on fpga", *IEEE Transactions on Biomedical Circuits and Systems*, vol. 17, no. 1, pp. 129–141, 2023.

[5] Q. Chen, D. Li, T. Tao, H. Ma, and E. Li, "Temporal neural encoding methods for spiking neural networks", in *2022 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC)*, IEEE, 2022, pp. 88–90.

[6] B. Petro, N. Kasabov, and R. M. Kiss, "Selection and optimization of temporal spike encoding methods for spiking neural networks", *IEEE transactions on neural networks and learning systems*, vol. 31, no. 2, pp. 358–370, 2019.

[7] S. Y. A. Yarga, J. Rouat, and S. Wood, "Efficient spike encoding algorithms for neuromorphic speech recognition", in *Proceedings of the International Conference on Neuromorphic Systems 2022*, 2022, pp. 1–8.

[8] A. Vasilache, *Alex-Vasilache/Spike-Encoding*, [Online]. Available: https://github.com/Alex-Vasilache/Spike-Encoding, Feb. 2025.

[9] A. Webb, S. Davies, and D. Lester, "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, pp. 259–267.

[10] B. Schrauwen and J. Van Campenhout, "Bsa, a fast and accurate spike train encoding scheme", in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, IEEE, vol. 4, 2003, pp. 2825–2830.

[11] N. Kasabov *et al.*, "Evolving spatio-temporal data machines based on the neucube neuromorphic framework: Design methodology and selected applications", *Neural Networks*, vol. 78, pp. 1–14, 2016.

[12] A. Arriandiaga, E. Portillo, J. I. Espinosa-Ramos, and N. K. Kasabov, "Pulsewidth modulation-based algorithm for spike phase encoding and decoding of time-dependent analog data", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 10, pp. 3920–3931, 2019.

[13] J. Dupeyroux, S. Stroobants, and G. C. De Croon, "A toolbox for neuromorphic perception in robotics", in *2022 8th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*, IEEE, 2022, pp. 1–7.

[14] A. R. Gollahalli, *Github.com/akshaybabloo/Spikes*, original-date: 2016-10-05, Sep. 2024.

[15] N. Kasabov *et al.*, "Evolving spatio-temporal data machines based on the neucube neuromorphic framework: Design methodology and selected applications", *Neural Networks*, vol. 78, pp. 1–14, 2016.

[16] B. Schrauwen and J. Van Campenhout, "Bsa, a fast and accurate spike train encoding scheme", in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, IEEE, vol. 4, 2003, pp. 2825–2830.

[17] A. Arriandiaga, E. Portillo, J. I. Espinosa-Ramos, and N. K. Kasabov, "Pulsewidth modulation-based algorithm for spike phase encoding and decoding of time-dependent analog data", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 10, pp. 3920–3931, 2019.

[18] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework", in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[19] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi, "Multiobjective tree-structured parzen estimator for computationally expensive optimization problems", in *Proceedings of the 2020 genetic and evolutionary computation conference*, 2020, pp. 533–541.

[20] M. Towers *et al.*, "Gymnasium: A standard interface for reinforcement learning environments", *arXiv preprint arXiv:2407.17032*, 2024.