Neural Networks 199 (2026) 108483

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

Zero-shot temporal resolution domain adaptation for spiking neural networks

Sanja Karilanova [a,*], Maxime Fabre [b,c], Emre Neftci [b], Ayça Özçelikkale [a]

[a] Uppsala University, Sweden
[b] RWTH and Forschungszentrum Jülich, Germany
[c] University of Groningen, Netherlands

ARTICLE INFO

Keywords:
Domain adaptation
Temporal resolution
Neuromorphic
Spiking neural networks (SNNs)
State-space models (SSMs)

ABSTRACT

Spiking Neural Networks (SNNs) are biologically-inspired deep neural networks that efficiently extract temporal information while offering promising gains in terms of energy efficiency and latency when deployed on neuromorphic devices. SNN parameters are sensitive to temporal resolution, leading to significant performance drops when the temporal resolution of target data during deployment is not the same as that of the source data used for training, especially when fine-tuning with the target data is not possible during deployment. To address this challenge, we propose three novel domain adaptation methods for adapting neuron parameters to account for the change in time resolution without re-training on target time resolution. The proposed methods are based on a mapping between neuron dynamics in SNNs and State Space Models (SSMs) and are applicable to general neuron models. We evaluate the proposed methods under spatio-temporal data tasks, namely the audio keyword spotting datasets SHD and MSWC, and the neuromorphic image NMINST dataset. Our methods provide an alternative to-and in most cases significantly outperform-the existing reference method that consists of scaling only the time constant. Notably, when the temporal resolution of the target data is double that of the source data, applying one of our proposed methods instead of the benchmark achieves classification accuracy of 89.5 % instead of 53.0 % on SHD, 93.6 % instead of 38.8 % on MSWC and 98.5 % instead of 97.2 % on NMNIST. Moreover, our results show that high accuracy on high temporal resolution data can be obtained by time-efficient training on lower temporal resolution data.

1. Introduction

Domain adaptation addresses the scenarios in which a model is confronted with a domain shift after deployment, while its task remains the same, such as face recognition under different lightning conditions or poses (Farahani et al., 2021; Liang et al., 2024). The domain shift refers to a change in feature space between the source domain on which the model was trained, and the target domain on which the model is evaluated. The change in feature space has been studied from various aspects, particularly for visual data (Patel et al., 2015; Zubic et al., 2024), but has rarely been explored with respect to temporal resolution shift for temporal data. Our work focuses precisely on this under-explored aspect, investigating feature discrepancies caused by differences in temporal resolution between the source and target data. While domain adaptation typically involves transferring knowledge between the source and target data by re-training parts of the model or the entire model using samples from target domain, Patel et al. (2015), Sun et al. (2019) and Bahi et al. (2025), we focus on the extreme case of no retraining on target domain.

In a conventional data acquisition set-up, where values of signals of interest are recorded at regular intervals, temporal resolution is primarily determined by the time interval between successive samples. For instance, the frame rate of a video stream or the sampling rate of an audio recording determines the temporal resolution. In the case of event-based sensing, e.g. Gallego et al. (2022), where changes in the scene are recorded with time-stamps, temporal resolution is determined by the temporal accuracy of time-stamps.

Various practical limitations lead to data with varying temporal resolution. For instance, limited memory capacity and power constraints often require lower sampling rates of data acquisition or storage devices (Dieter et al., 2005; ur Rehman et al., 2016). Additionally, sensor quality and bandwidth limitations further restrict temporal resolution, as high-resolution sensors and real-time transmission demand more resources (Chen et al., 2021; Park et al., 2021). Consequently, changes

* Corresponding author.
E-mail addresses: sanja.karilanova@angstrom.uu.se (S. Karilanova), m.p.y.fabre@rug.nl (M. Fabre), e.neftci@fz-juelich.de (E. Neftci), ayca.ozcelikkale@angstrom.uu.se (A. Özçelikkale).

https://doi.org/10.1016/j.neunet.2025.108483
Received 2 June 2025; Received in revised form 4 November 2025; Accepted 13 December 2025
Available online 20 December 2025
0893-6080/© 2025 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

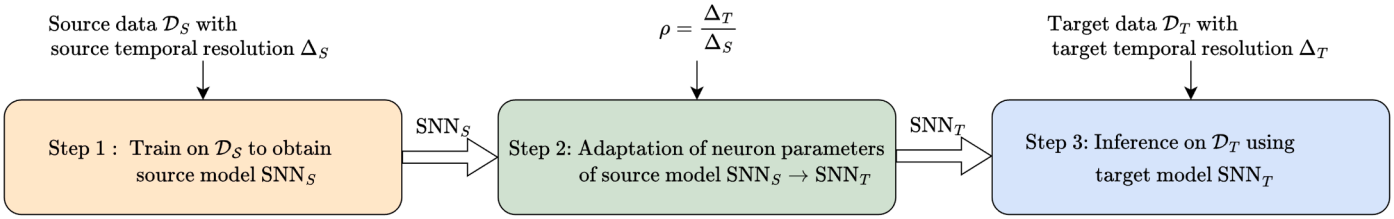


Fig. 1. Overview of the set-up. We investigate how models trained on data with a given source temporal resolution can be adapted for data with a different target temporal resolution, both in the Fine-to-Coarse and Coarse-to-Fine deployment directions.

in the temporal resolution of data have been the focus of a wide range of studies, such as increasing of temporal resolution of videos (Zuckerman et al., 2020), optimization of sensor networks' sensing rates (Jain & Chang, 2004), processing of irregularly sampled data (Rubanova et al., 2019), optimal combination of high- and low-rate time-series data (Bae et al., 2024). In the case of event-based cameras, effect of bandwidth and event-rate constraints on the temporal resolution have been studied (Gehrig & Scaramuzza, 2022; Zubic et al., 2024).

In this paper, we are interested in the scenario where the temporal resolution of the source data which the model has been trained on, and the target data which the model is to be used on is different. We investigate both the scenario of Fine-to-Coarse deployment, where the change is from high to low resolution, and Coarse-to-Fine deployment, where the change is from low to high resolution. We consider the setting where it is not possible to fine-tune the model at the edge on the data with the target temporal resolution, for instance when real-time inference is needed on the current sample from target domain. Hence, we consider temporal resolution domain adaptation with no re-training on the target dataset, see Fig. 1.

Applications of these scenarios appear in different edge processing scenarios, where small resource-constrained devices with different sensing and computing capabilities must process their data at the edge of a wireless system with varying degrees of support from a central processing or coordination unit (Chen et al., 2021; Park et al., 2021). A possible application of Coarse-to-Fine deployment is encountered in communication constrained distributed scenarios with a central learning unit. In such scenarios, sensor data is sent to a central learning unit with reduced temporal resolution due to bandwidth or power limitations (Chen et al., 2021; Park et al., 2021). As a result, the central model is trained on coarse data and then deployed to edge devices, where higher-resolution data is available for inference. A possible application of Fine-to-Coarse deployment is encountered where the sensors have to switch to low sampling rates to reduce power consumption (Dieter et al., 2005; Jain & Chang, 2004). This leads to having target data at edge with coarser temporal resolution than the source data used for training the deployed models at the central learning unit.

Applications of these scenarios appear in different edge processing scenarios, where small resource-constrained devices with different sensing and computing capabilities must process their data at the edge of a wireless system with varying degrees of support from a central processing or coordination unit (Chen et al., 2021; Park et al., 2021). A possible application of Coarse-to-Fine deployment is encountered in communication constrained distributed scenarios with a central learning unit. In such scenarios, sensor data is sent to a central learning unit with reduced temporal resolution due to bandwidth or power limitations (Chen et al., 2021; Park et al., 2021). As a result, the central model is trained on coarse data and then deployed to edge devices, where higher-resolution data is available for inference. A possible application of Fine-to-Coarse deployment is encountered where the sensors have to switch to low sampling rates to reduce power consumption (Dieter et al., 2005; Jain & Chang, 2004). This leads to having target data at edge with coarser temporal resolution than the source data used for training the deployed models at the central learning unit.

We consider the above temporal domain adaptation problems under Spiking Neural Networks (SNN) models. SNNs are biologically-inspired neural networks with stateful recurrent neurons, and binary spike-based communication (Maass, 1997). SNNs provide an attractive framework for processing data with encoded temporal information due to their ability to represent and learn spatiotemporal features (Bittar & Garner, 2022; Eshraghian et al., 2023; Ma et al., 2025; Müller-Cleve et al., 2022; Yik, 2025). Like most machine learning algorithms, the current mainstream pipeline for SNNs relies on training at high-power computational units, such as GPUs, and then deploying and running on low-power edge devices (Abadade et al., 2023). SNNs, when deployed at neuromorphic edge hardware, are especially valuable because they are designed to mimic the brain's energy-efficient processes, offering significant reductions in power consumption and latency (Caccavella et al., 2024; Davies et al., 2021; Müller-Cleve et al., 2022). SNNs are particularly effective in event-based systems, where event-based sensors, e.g. event-cameras, record changes only as they occur, rather than at fixed time intervals, leading to optimized energy use and reduced data redundancy (Gallego et al., 2022).

A common practical training set-up for SNNs is the standard digital clock-based SNNs, where the temporal resolution of the data corresponds directly to the processing resolution at which the model operates. In this setup, the possible discrepancies in the temporal resolution of the source and target data constitute a significant challenge since trained SNN model parameters are strongly dependent on the temporal resolution of the source data. Hence, an SNN model that has been trained using data with a given temporal resolution may not necessarily perform well in inference on data from another sensory source even if it is recording of the same phenomena, e.g., see Section 5.

Training on sequences with high temporal resolution is desired due to the possible inclusion of more information and therefore better performance of the model (Gehrig & Scaramuzza, 2024). However, this comes with a cost. Difficulties arise when training SNNs on long sequences such as exploding and vanishing gradients and other instabilities due to the recurrent and saturating nature of SNNs (Zucchet & Orvieto, 2024). Another limitation while training on long sequences is the significant training time as the temporal resolution dimension approximately linearly increases the training time and computational complexity (Perez-Nieves & Goodman, 2021). Hence, the development of temporal adaptation methods enables training models in a fast and stable manner on low temporal resolution data, while still ensuring high performance during inference on high temporal resolution data. This setting supports energy, time, and computation efficient training pipeline.

Considering both the temporal domain adaptation problem as well as challenges of training on sequences with fine temporal resolution, the main contributions of this paper are:

- We propose three novel methods for adapting SNN parameters that govern the neuron dynamics for changes in temporal resolution of the data without using any data samples with new temporal resolution.

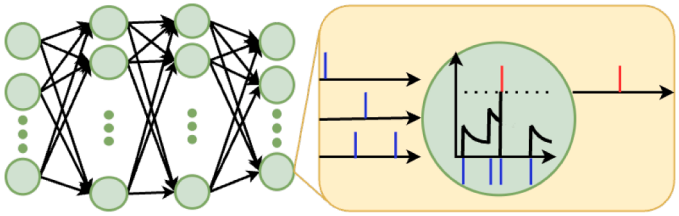


Fig. 2. Feedforward SNN model architecture consisting of four layers, each with arbitrary number of neurons. An example neuron dynamics given as a zoom-in on a neuron.

- The performance of the proposed methods is demonstrated using the audio datasets SHD (Cramer et al., 2022) and MSWC (Mazumder et al., 2024) and the vision NMNIST dataset (Orchard et al., 2015).
- The results show that in a wide range of scenarios, e.g. coarse-to-fine scenarios for all the above datasets, our proposed methods provide significant performance gains compared to using no temporal resolution domain adaptation method or using the existing benchmark method from the literature.
- We show that utilizing version of the data with low temporal resolution for pre-training, constitutes a promising training approach for improving computational training efficiency in SNNs.

We now provide an overview of the organization of the paper. Section 2 provides a review of the related work. The main problem studied in this article, i.e., the temporal domain adaptation problem, is formulated in Section 3. Section 4 presents the proposed methods. Numerical results illustrating the performance of our methods are provided in Sections 5 and 6. Finally, in Sections 7 and 8 we present further discussions and a summary of our conclusions, respectively.

2. Related work

The Leaky-Integrate-and-Fire (LIF) neuron model is one of the most simple and widely used neuron models in the context of SNNs. When input stimuli in the form of spikes are applied to the neuron, the membrane potential increases and when a certain threshold is reached it produces an output spike. On the other hand, in the absence of input stimuli, the membrane potential decays exponentially (Bittar & Garner, 2022), see Fig. 2. The vanilla LIF neuron has only one learnable parameter, which is the membrane potential decay (the leak), and this parameter has an explicit exponential dependence on the time step length of the data. Hence, when dealing with a change of temporal resolution, intuitive way to adapt the LIF neuron dynamic is to scale the leak parameter according to the change. In He et al. (2020),Table 12 the authors explore this method of temporal domain adaptation using the simple LIF neuron. On the other hand, in Caccavella et al. (2024), the authors use Integrate-and-Fire (IF) neuron which is a special simpler case of LIF neuron without leak. Hence alternative to parameters adaptation, they employ a graded spike network and different normalization techniques for domain adaptation with promising results. However, in order to capture more complex temporal information in data, more complex neuron models such as the adaptive LIF (adLIF) neurons have shown superior performance (Baronig et al., 2025; Bittar & Garner, 2022; Ma et al., 2025; Yin et al., 2021). In the case of more complex neuron models however, the parameter might not have explicit dependence on the time-length of the data, or even any dependence at all.

In this case, it is unclear how these parameters need to be scaled to depict the change of temporal resolution, and to the best of our knowledge, this issue has not been thoroughly addressed in the existing literature.

An important building block of our proposed approach is State Space Models (SSMs), which is a general framework for modeling dynamic systems. Their neural network implementations, i.e. deep SSMs, raised as an attractive alternative to the conventional deep recurrent models. It was shown that for appropriate conditioning on the state matrix dynamics, deep SSMs can handle long-range dependencies while offering stable and computationally efficient training (Wang et al., 2024). Limited explorations of temporal resolution change have been performed using SSMs and traditional data [Smith et al., 2022, Table 2]. Deep SSMs have been used for event-based data in a number of works (Schöne et al., 2024; Zubic et al., 2024) Similar to the leak adaptation in LIF neurons, explicit correlation can be addressed between significantly simplified version of SSM and the temporal dynamics between spikes. Schöne et al. (2024) explores this concept by processing incoming spikes with adapted leak based on the time-length between consecutive spikes. Alternatively, Zubic et al. (2024) explored varying-resolution aware training of deep SSMs for vision dataset, where the training includes masking inputs outside specific frequency ranges and incorporating an additional loss function to penalize high-frequency activities.

Interesting intersection of work is emerging between SNNs and deep SSMs. One common line of work is embedding spiking dynamics in deep SSMs where at each time step the output activity of the SSM is processed by a firing threshold function (Bal & Sengupta, 2025; Karilanova et al., 2025a,b; Vincent-Lamarre et al., 2020). Alternatively, in Du et al. (2024) they combine SNN with SSMs by adding a spiking layer to the SSM. Nevertheless, the challenge of drawing lines of correspondence between SNNs and linear SSMs is still present due to the non-linearity when it comes to SNNs. Our proposed methods for temporal domain adaptation for SNNs are based on this drawn parallel and the theory developed for adaptation in SSMs. See Table 1 for a summary of the prior works discussed in this section.

3. Problem statement

In Section 3.1 we present a popular neuron model commonly used in the SNN literature to establish a foundation for the generalized neuron model in Section 3.2. We outline the temporal domain adaptation problem that this article focuses on in Section 3.3.

3.1. Popular spiking neuron model

The discrete-time adaptive Leaky-Integrate and Fire (adLIF) neuron is defined as (Bittar & Garner, 2022)

$$u[t+1] = \alpha(u[t] - \theta s_{out}[t]) + (1-\alpha)(\boldsymbol{W}\boldsymbol{s}_{in}[t] + \boldsymbol{V}\boldsymbol{s}_{out}[t]) - (1-\alpha)u[t] \tag{1a}$$

$$u[t+1] = au[t] + \beta u[t] + bs_{out}[t] \tag{1b}$$

$$s_{out}[t] = g_\theta(u[t]) \tag{1c}$$

where $\mathbf{s}_{in} \in \mathbb{Z}^{L_b \times 1}$ are the input spikes from the neurons in the previous layer and $\mathbf{W} \in \mathbb{R}^{1 \times L_b}$ is the weight matrix associated with their connections to the neuron of observation, while $s_{out} \in \mathbb{Z}^{L_c \times 1}$ are the output spikes of all the other neurons in the current layer and $\boldsymbol{V} \in \mathbb{R}^{1 \times L_c}$ with zero diagonal is their connections to the neuron of observation. Here, L_c and L_b are the number of neurons in the current and the previous layer, respectively. Furthermore, $g_\theta(\cdot)$ describes the spiking behavior with the firing threshold $\theta \in \mathbb{R}^{1\times 1}$; $\alpha = \exp(\Delta/\tau_{mem})$ represents the decay of the membrane potential over time such that Δ corresponds to the duration of a timestep and τ_{mem} is a membrane time constant; $\beta = \exp(\Delta/\tau_{syn})$ represents the decay of the adaptive variable with adaptation constant τ_{syn}; the parameters a and b are without explicit dependence on the discrete time step length Δ. All parameters α, β, a, b are generally learnable during training within given ranges.

Note, the weight vector $\boldsymbol{V}$ corresponds to the recurrent connections between the neurons in the same layer. Recurrence represented by $\boldsymbol{V}$ is a characteristic of the network rather than the neuron.

Table 1
Overview of works which include aspects of zero-shot temporal adaptation in SNN or deep-SSM models.

Method	Neuron model	Adaptation	Datasets	Metrics
This article	Generalized spiking neuron including adLIF, LIF, IF, SSM	Expectation, Integral, and Euler Methods	SHD, MSWC, N-MNIST	Classification accuracy in varying temoral resolution
Comparing SNNs and ANNs (He et al., 2020),Table 12	LIF, rate-coded RNN and rate-coded LSTM	No-adaptation	N-MNIST	Classification accuracy in varying temporal resolution
Event-driven inference (Caccavella et al., 2024)	IF	Layer normalization	class "faces" of N-CALTECH101	mAP on varying time-window of integration
EventSSM (Schöne et al., 2024)	SSM	Event-by-event processing	SHD, SSC, DVS-Gesture	Classification accuracy (both in train and test stage event-by-event processing used)
SSM for event cameras (Zubic et al., 2024)	SSM	Low-pass bandlimiting, and SSM-ViT block	Gen1 and 1 Mpx	mAP on varying evaluations frequency

3.2. General spiking neuron model

In this section, we introduce a discrete-time general n-dimensional spiking neuron model, which generalizes the two-dimensional adLIF neuron (1). This generalized model is central to the proposed methods of this paper. It is defined as

$$\boldsymbol{V}[t+1] = \boldsymbol{H}_v \boldsymbol{v}[t] + \boldsymbol{H}_f s_{out}[t] + \boldsymbol{H}_i \boldsymbol{W} \boldsymbol{s}_{in}[t] + \boldsymbol{H}_r \boldsymbol{V} \boldsymbol{s}_{out}[t] \tag{2a}$$

$$s_{out}[t] = g_{\boldsymbol{\Theta}}(\boldsymbol{V}[t]) = \begin{cases} 1 \text{ if } \boldsymbol{v}[t] \in \boldsymbol{\Theta} \\ 0 \text{ otherwise} \end{cases} \tag{2b}$$

where $\boldsymbol{v} \in \mathbb{R}^{n\times 1}$ is the state variable, while $\boldsymbol{H}_v \in \mathbb{R}^{n\times n}$ and $\boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r \in \mathbb{R}^{n\times 1}$ are matrices containing the parameters of the individual neuron. The subscripts of the $\boldsymbol{H}$ matrices, i.e. v, f, i, and r, indicate state, feedback, input and recurrence, respectively. The spiking behavior is described by $g_{\boldsymbol{\Theta}}(\cdot)$, where the neuron spikes when the state enters in the region described by $\boldsymbol{\Theta}$. The region $\boldsymbol{\Theta}$ depends on the neuron model.

Many popular spiking neuron models can be expressed as special cases of the general spiking neuron model in (2) including the LIF neuron (Gerstner et al., 2014) with $n = 1$ state variable, the adLIF neuron (Bittar & Garner, 2022) with $n = 2$, the Izikevich neuron (Izhikevich, 2003) with $n = 2$, the Hodgkin-Huxley (Hodgkin & Huxley, 1952) with $n = 4$, multi-compartment neurons with 2 compartments each with $n = 2$ (Zhang et al., 2024). Below, we show how the $n = 2$ dimensional adLIF neuron can be expressed as a special case of (2).

The adLIF neuron, in matrix form, can be written as:

$$\begin{bmatrix} u[t+1] \\ u[t+1] \end{bmatrix} = \begin{bmatrix} \alpha & -(1-\alpha) \\ a & \beta \end{bmatrix} \begin{bmatrix} u[t] \\ u[t] \end{bmatrix} + \begin{bmatrix} -\alpha\theta \\ b \end{bmatrix} s_{out}[t] + \begin{bmatrix} 1-\alpha \\ 0 \end{bmatrix} \boldsymbol{W} \boldsymbol{s}_{in}[t] + \begin{bmatrix} 1-\alpha \\ 0 \end{bmatrix} \boldsymbol{V} \boldsymbol{s}_{out}[t]. \tag{3}$$

Thus, the adLIF neuron (1) with two state variables $\boldsymbol{v} = \left[uNoArg, uNoArg\right]^T$ is a special case of the general model in (2) where

$$\boldsymbol{H}_v = \begin{bmatrix} \alpha & -(1-\alpha) \\ a & \beta \end{bmatrix}, \boldsymbol{H}_f = \begin{bmatrix} -\alpha\theta \\ b \end{bmatrix}, \boldsymbol{H}_i = \begin{bmatrix} 1-\alpha \\ 0 \end{bmatrix}, \text{and } \boldsymbol{H}_r = \begin{bmatrix} 1-\alpha \\ 0 \end{bmatrix}. \tag{4}$$

Remark 1. The general spiking neuron model in (2) does not encompass all possible neuron formulations. For instance, it does not capture: (i) hard reset mechanism, where the neuron state is set to a predefined reset state value when spiking occurs (Davies et al., 2018) rather then subtracting the spike using $\boldsymbol{H}_f$; (ii) varying adaptive spiking threshold, where the threshold varies over time (Chen et al., 2022), rather than being defined by a fixed spiking region $\boldsymbol{\Theta}$.

3.3. Problem formulation - temporal resolution domain adaptation for general spiking neuron model

3.3.1. Event-based data

In this section, we state the event-based data representation we focus on in this article. Event-based data can be represented as asynchronous or synchronous (Chakravarthi et al., 2024; Gallego et al., 2022). In the asynchronous representation, data is provided as a list of events each with their precise time of occurrence. In the synchronous representation, data is provided in a time-binned format as spike sequences of a given length, each bin populated with all events occurring during this period of time. The first type of representation is naturally adopted when data comes from event-based sensors, such as NMNIST (Orchard et al., 2015) and the second type of data representation is typically used after post-processing of data obtained by conventional sensors, such as Spiking Heidelberg Digits (SHD) dataset (Cramer et al., 2022), or by post-processing of data from event-based sensors.

The standard approach for processing event-based data with SNNs is to use digital clock-based SNNs, which require the data to be in an effectively synchronous form for processing. Hence, if the data is asynchronous, it is converted into a synchronous form, such as by binning events on regular time intervals. In this article, we focus on such synchronous data representations where events that occur during a temporal window of fixed length are accumulated to create sequences of spikes (Gallego et al., 2022; Liu & Delbruck, 2018). This type of representations have been used in a large number of earlier SNN works, including studies utilizing temporal bins for audio signals (Bittar & Garner, 2022; Hammouamri et al., 2024) and frame-based event representation for spatio-temporal data (Caccavella et al., 2024; He et al., 2020; Zubic et al., 2024).

3.3.2. Temporal resolution

In this section, we define the notion of temporal resolution used in this article. Consider a continuous-time signal $\boldsymbol{s}(t), 0 \le t \le t_c$ with a duration t_c seconds, representing a phenomenon of interest, such as a spoken keyword. We denote by $\boldsymbol{s} \in \mathbb{Z}^{N\times 1}$ an integer-valued discrete-time sequence extrapolated from $\boldsymbol{s}(t)$ where N is the number of time steps. Hence, for a regular discretization of the time-axis, which we focus on here, a single time step in $\boldsymbol{s}$ will represent a time duration of $\Delta_N = t_c/N$ seconds.

Suppose that we have two discrete-time data sequences associated with the same continuous signal, each with a different number of time steps. Let us refer to them as $\boldsymbol{s}_S \in \mathbb{Z}^{N_S\times 1}$ and $\boldsymbol{s}_T \in \mathbb{Z}^{N_T\times 1}$ with $N_S \neq N_T$, and thus $\Delta_S \neq \Delta_T$. Hence, $\boldsymbol{s}_S$ and $\boldsymbol{s}_T$ correspond to sequences with different time resolutions. The sequences $\boldsymbol{s}_S$ and $\boldsymbol{s}_T$ can be obtained from $\boldsymbol{s}(t)$ in various ways. One is to obtain both sequences directly from

$\boldsymbol{s}(t)$ using different encoding parameters. Another is to obtain $\boldsymbol{s}_S$ directly from $\boldsymbol{s}(t)$, while obtaining $\boldsymbol{s}_T$ by transforming $\boldsymbol{s}_S$, or vice versa.

3.3.3. Model adaptation

We consider the scenario where the SNN model is pre-trained on data with time resolution Δ_S but we would like to use this model on data with temporal resolution Δ_T, where $\Delta_T \neq \Delta_S$. Consequently, we refer to the training dataset as source data and the latter as target data. Similarly, we refer to Δ_S and Δ_T as the temporal resolution of the source and target data, respectively.

During SNN training, two groups of parameters are learned: parameters of the neuron dynamics for each neuron, $\boldsymbol{H}_{v,(S)}, \boldsymbol{H}_{f,(S)}, \boldsymbol{H}_{i,(S)}, \boldsymbol{H}_{r,(S)}$, and the weights of the synaptic connections between the neurons, $\mathbf{W}, \boldsymbol{V}$. Our main goal in this article is to adjust the model parameters such that we obtain high-accuracy results on target data. We propose to achieve this goal by keeping the dynamics of the state variable $\boldsymbol{V}$ of neurons, see (2), as similar as possible under the source and target data. This means that the dynamics of the state variable $\boldsymbol{V}$ is preserved even when the temporal resolution of the data changes.

We consider this problem under two constraints: First, we are only allowed to adapt the neuron's dynamic parameters, $\boldsymbol{H}_k$ for $k = v, i, f, r$. Hence, SNN architecture and synaptic connections are unchanged. Second, the only known information about the target data is the ratio between the target and the source temporal resolution. Hence, we do not rely on any post-deployment re-training on the target dataset, which can be costly. Our results in Section 5 show that promising performance can be obtained even under this constrained scenario, supporting the central role of neuron dynamics in capturing time-dependent behavior.

In summary, the main goal of this article is to find a mapping from the source to the target feature space in the form of

$$\begin{aligned}&\boldsymbol{H}_{v,(T)}, \boldsymbol{H}_{i,(T)}, \boldsymbol{H}_{f,(T)}, \boldsymbol{H}_{r,(T)}\\ &\quad = \mathcal{M}(\rho, \boldsymbol{H}_{v,(S)}, \boldsymbol{H}_{i,(S)}, \boldsymbol{H}_{f,(S)}, \boldsymbol{H}_{r,(S)}),\end{aligned} \tag{5}$$

where the mapping $\mathcal{M}(.)$ represents the adaptation of the temporal resolution domain, and the parameter ρ is given by

$$\rho = \frac{\Delta_T}{\Delta_S} = \frac{N_S}{N_T} = \frac{1}{\bar{\rho}} \tag{6}$$

defining the ratio between the temporal resolutions. Here, we have also defined $\bar{\rho}$, the reciprocal of ρ, for a more convenient notation in later chapters. We consider the following scenarios:

- Coarse-to-Fine deployment: The model is pre-trained on source data with coarse temporal resolution and tested on target data with fine temporal resolution, i.e., $\Delta_S > \Delta_T$, thus $\rho < 1$.
- Fine-to-Coarse deployment: The model is pre-trained on source data with fine temporal resolution and tested on target data with coarse temporal resolution, i.e., $\Delta_S < \Delta_T$, thus $\rho > 1$.

4. Methods

As a background for our work, in Section 4.1, we begin by expressing the SNN neuron model as a non-linear SSM model. This provides a foundation for Section 4.2, where we propose an approximate correspondence between the SNN neuron model and linear SSM model. The proposed temporal resolution domain adaptation methods, presented in Section 4.3, are based on this approximate correspondence. In Section 4.4 we present a simple benchmark method.

4.1. Preliminaries - correspondence between SNNs and non-linear SSMs

In this section, we express the SNN neuron model as a specific case of non-linear SSMs.

A general discrete-time nonlinear SSM can be written as (Okuyama, 2014):

$$\boldsymbol{v}[t+1] = \mathcal{F}(\boldsymbol{v}[t], \boldsymbol{f}[t]) \tag{7a}$$

$$\boldsymbol{y}[t] = \mathcal{G}(\boldsymbol{v}[t], \boldsymbol{f}[t]) \tag{7b}$$

where $\boldsymbol{v}[t] \in \mathbb{R}^{n\times 1}$, $\boldsymbol{f}[t] \in \mathbb{R}^{r\times 1}$, $\boldsymbol{y}[t] \in \mathbb{R}^{p\times 1}$ represent the state vector, the input to the system and the output of the system, respectively, and $\mathcal{F}(\cdot)$ and $\mathcal{G}(\cdot)$ are possibly non-linear functions. Here, n is the dimension of the system's state, r is the dimension of the system's input, and p is the dimension of the system's output. The general neuron model in (2) can be expressed as a non-linear SSM as in (7) using

$$\mathcal{F}(\boldsymbol{v}[t], \boldsymbol{f}[t]) = \boldsymbol{H}_v \boldsymbol{v}[t] + \boldsymbol{H}_f g_{\boldsymbol{\Theta}}(\boldsymbol{v}[t]) + \left[\boldsymbol{H}_i, \boldsymbol{H}_r\right] \boldsymbol{f}[t] \tag{8a}$$

$$\mathcal{G}(\boldsymbol{v}[t], \boldsymbol{f}[t]) = g_{\boldsymbol{\Theta}}(\boldsymbol{v}[t]) \tag{8b}$$

where $\boldsymbol{f}[t] = \left[\boldsymbol{W}\boldsymbol{s}_{in}[t], \boldsymbol{V}\boldsymbol{s}_{out}[t]\right]^T$ is considered external input since $\boldsymbol{s}_{in}[t]$ parameterize the spikes from the previous layer and $\boldsymbol{s}_{out}[t]$ parameterize the spikes from all the other neurons in the current layer. Here, $\boldsymbol{y}$ represents the output spike of the neuron.

Since our ultimate goal is to find a correspondence between SNNs and linear SSMs, we move one step closer to a linear SSM by considering separable $\mathcal{F}(\cdot)$ and $\mathcal{G}(\cdot)$, i.e.,

$$\boldsymbol{v}[t+1] = \mathcal{F}_A(\boldsymbol{v}[t]) + \mathcal{F}_B(\boldsymbol{f}[t]) \tag{9a}$$

$$\boldsymbol{y}[t] = \mathcal{G}_C(\boldsymbol{v}[t]) + \mathcal{G}_D(\boldsymbol{f}[t]) \tag{9b}$$

where $\mathcal{F}_A(\cdot), \mathcal{F}_B(\cdot), \mathcal{G}_C(\cdot)$ and $\mathcal{G}_D(\cdot)$ are possibly non-linear functions. The generalized neuron in (2) can be expressed as a non-linear SSM with separable functions as in (9) using

$$\mathcal{F}_A(\boldsymbol{v}[t]) = \boldsymbol{H}_v \boldsymbol{v}[t] + \boldsymbol{H}_f g_{\boldsymbol{\Theta}}(\boldsymbol{v}[t]) \tag{10a}$$

$$\mathcal{F}_B(\boldsymbol{f}[t]) = \left[\boldsymbol{H}_i, \boldsymbol{H}_r\right] \boldsymbol{f}[t] \tag{10b}$$

$$\mathcal{G}_C(\boldsymbol{v}[t]) = g_{\boldsymbol{\Theta}}(\boldsymbol{v}[t]) \tag{10c}$$

$$\mathcal{G}_D(\boldsymbol{f}[t]) = \mathbf{0} \tag{10d}$$

where $\boldsymbol{f}[t] = \left[\boldsymbol{W}\boldsymbol{s}_{in}[t], \boldsymbol{V}\boldsymbol{s}_{out}[t]\right]^T$.

4.2. Preliminaries - correspondence between SNNs and linear SSMs

As linear SSMs enable analytical investigations and form the basis of our adaptation methods, in this section, we propose an approximate correspondence between the general SNN neuron model in (2) and linear SSMs. Despite SNNs being strictly non-linear SSMs, this approximate correspondence can be utilized to achieve promising results for temporal domain adaptation, as demonstrated by the results in Section 5.

A linear SSM in discrete-time can be written as (Gajic, 2003):

$$\boldsymbol{u}[t+1] = \boldsymbol{A}\boldsymbol{u}[t] + \boldsymbol{B}\boldsymbol{f}[t] \tag{11a}$$

$$\boldsymbol{y}[t] = \boldsymbol{C}\boldsymbol{u}[t] + \boldsymbol{D}\boldsymbol{f}[t], \tag{11b}$$

where the vectors $\boldsymbol{v}[t], \boldsymbol{f}[t], \boldsymbol{y}[t]$ are respectively the state vector, input vector and output vector. Here, the matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ describes the internal behaviour of the system, while matrices $\boldsymbol{B} \in \mathbb{R}^{n\times r}, \boldsymbol{C} \in \mathbb{R}^{p\times n}, \boldsymbol{D} \in \mathbb{R}^{p\times r}$ represent connections between the external world and the system. These matrices represent arrays of real scalar numbers. The system is assumed to be time invariant which means the scalar numbers are constant over time.

To find an approximate correspondence between the general neuron model in (2) and the linear SSM in (11), we first map the nonlinear function in the separable non-linear SSM (9) to the matrices describing linear relation in (11). The mapping is intuitive,

$$\boldsymbol{A}\boldsymbol{v}[t] \rightarrow \mathcal{F}_A(\boldsymbol{v}[t]) \tag{12a}$$

$$\boldsymbol{B}\boldsymbol{f}[t] \rightarrow \mathcal{F}_B(\boldsymbol{f}[t]) \tag{12b}$$

$$\boldsymbol{C}\boldsymbol{v}[t] \rightarrow \mathcal{G}_C(\boldsymbol{v}[t]) \tag{12c}$$

$$\boldsymbol{D}\boldsymbol{f}[t] \rightarrow \mathcal{G}_D(\boldsymbol{f}[t]) \tag{12d}$$

where the symbol $\rightarrow$ indicates that the left-hand side linear term is best described as part of the dynamics of the right-hand side possibly nonlinear function. Next, we associate the general neuron matrices $\boldsymbol{H}_v, \boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r$ in (2), to the linear matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ in (11). Using

Algorithm 1 Pseudo-code for the work-flow with the proposed adaptation methods.

(1) Train on source resolution

Input : Training dataset $\mathcal{D}_S$ with source resolution Δ_S.

Initialize model randomly $SNN_S = (\boldsymbol{H}^{(0)}_{v,(S)}, \boldsymbol{H}^{(0)}_{i,(S)}, \boldsymbol{H}^{(0)}_{f,(S)}, \boldsymbol{H}^{(0)}_{r,(S)}, \boldsymbol{W}^{(0)}_S, \boldsymbol{V}^{(0)}_S)$.

for *epoch* $= 1, \ldots, E$ **do**

  Update the SNN_S model parameters using $\mathcal{D}_S$.

return $SNN_S = (\boldsymbol{H}_{v,(S)}, \boldsymbol{H}_{i,(S)}, \boldsymbol{H}_{f,(S)}, \boldsymbol{H}_{r,(S)}, \boldsymbol{W}_S, \boldsymbol{V}_S)$.

(2) Perform adaptation

Input : Trained SNN_S, ρ, i.e., the ratio $\frac{\Delta_T}{\Delta_S}$, choice of adaptation method, i.e., Integral, Euler or Expectation.

Adapt neuron model parameters of SNN_S i.e. set

$\boldsymbol{H}_{v,(T)}\boldsymbol{H}_{i,(T)}, \boldsymbol{H}_{f,(T)}, \boldsymbol{H}_{r,(T)} = \mathcal{M}(\rho, \boldsymbol{H}_{v,(S)}, \boldsymbol{H}_{i,(S)}, \boldsymbol{H}_{f,(S)}, \boldsymbol{H}_{r,(S)})$ using (16) for Integral (17) for Euler, or (18) for Expectation.

Create the model $SNN_T = (\boldsymbol{H}_{v,(T)}, \boldsymbol{H}_{i,(T)}, \boldsymbol{H}_{f,(T)}, \boldsymbol{H}_{r,(T)}, \boldsymbol{W}_S, \boldsymbol{V}_S)$.

return SNN_T.

(3) Use on data with target resolution

Input : SNN_T, testing dataset $\mathcal{D}_T$ with target resolution Δ_T.

Output : Performance of SNN_T on $\mathcal{D}_T$ i.e., $SNN_T(\mathcal{D}_T)$.

the correspondences in (10) and (12), the following mapping arises as a candidate: $\boldsymbol{A} = \boldsymbol{H}_v + \boldsymbol{H}_f \boldsymbol{G}_{\boldsymbol{\Theta}}$ and $\boldsymbol{B} = [\boldsymbol{H}_i, \boldsymbol{H}_r]$, where $\boldsymbol{G}_{\boldsymbol{\Theta}} \in \mathbb{R}^{1\times n}$ represents a linearization of the non-linear spiking function $g_{\boldsymbol{\Theta}}(\cdot)$. However, we propose to make a deviation from this mapping and interpret the spike of the neuron itself as an external input to the neuron state. This deviation avoids the linearization of the non-linear spiking function. Hence, we use $\boldsymbol{A} = \boldsymbol{H}_v$, and $\boldsymbol{B} = [\boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r]$. To visualize this correspondence, we rewrite the general spiking neuron in (2) as

$$\boldsymbol{v}[t+1] = \boldsymbol{H}_v \boldsymbol{v}[t] + [\boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r] \begin{bmatrix} s_{out}[t] \\ \mathbf{W}\boldsymbol{s}_{in}[t] \\ \boldsymbol{v}\boldsymbol{s}_{out}[t] \end{bmatrix} \tag{13a}$$

$$s_{out}[t] = g_{\boldsymbol{\Theta}}(\boldsymbol{v}[t]) \tag{13b}$$

Hence, we propose the following approximate correspondence between the general SNN neuron model and the linear SSM framework:

$$\boldsymbol{v}[t+1] = \boldsymbol{A}\boldsymbol{v}[t] + \boldsymbol{B}\boldsymbol{f}[t] \tag{14}$$

where $\boldsymbol{v} \in \mathbb{R}^{n\times 1}$, $\boldsymbol{A} = \boldsymbol{H}_v \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} = [\boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r] \in \mathbb{R}^{n\times r}$, and $\boldsymbol{f}[t] = [s_{out}[t], \mathbf{W}\boldsymbol{s}_{in}[t], \boldsymbol{V}\boldsymbol{s}_{out}[t]]^T \in \mathbb{R}^{r\times 1}$ such that $\boldsymbol{f}$ combines all input spikes to the neuron, including the effect of its own output. Since only the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices are needed for adapting the state variable dynamics, which is our main interest here, we leave (10c) and (10d) as they are, since these functions are associated with the output spikes $\boldsymbol{y}[t]$ and are not required for adaptation of state dynamics.

We now consider adLIF neuron as an example. For adLIF, using (4), we obtain the following as $\boldsymbol{A}$ and $\boldsymbol{B}$ of (14):

$$\boldsymbol{A} = \boldsymbol{H}_v = \begin{bmatrix} \alpha & -(1-\alpha) \\ a & \beta \end{bmatrix}, \tag{15a}$$

$$\boldsymbol{B} = [\boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r] = \begin{bmatrix} -\alpha\theta & 1-\alpha & 1-\alpha \\ b & 0 & 0 \end{bmatrix}. \tag{15b}$$

4.3. Proposed methods for temporal resolution domain adaptation

In this section, we define our proposed temporal resolution domain adaptation methods for the generalized SNN neuron model in (2). As explained in Section 3.3, the proposed methods suggest a mapping $\mathcal{M}(.)$ from the source domain with temporal resolution Δ_S to the target domain with temporal resolution Δ_T, where $\rho = \frac{\Delta_T}{\Delta_S}$.

Note that all the methods presented below have been derived in detail in the form of propositions in Section Appendix A.1. A step-by-step pseudo-code outlining the usage of the methods is provided in Algorithm 1.

Integral approximation adaptation method is derived from the Integral approximation method for obtaining discrete-time SSM from a continuous-time system model. Following Proposition 1 in Section Appendix A.1, the proposed mapping is

$$\boldsymbol{H}_v T = \left(\boldsymbol{H}_{v,(S)}\right)^{\rho} \tag{16a}$$

$$\boldsymbol{H}_{k,(T)} = (\boldsymbol{H}_{v,(T)} - \boldsymbol{I})(\boldsymbol{H}_{v,(S)} - \boldsymbol{I})^{-1}\boldsymbol{H}_{k,(S)} \tag{16b}$$

for $k = s, f, r$.

Euler adaptation method is derived from the Euler approximation method for obtaining discrete-time SSM from a continuous-time system model. Following Proposition 2 in Section Appendix A.1, the proposed mapping is

$$\boldsymbol{H}_{v,(T)} = \boldsymbol{I} + \rho(\boldsymbol{H}_{v,(S)} - \boldsymbol{I}) \tag{17a}$$

$$\boldsymbol{H}_{k,(T)} = \rho\boldsymbol{H}_{k,(T)} \tag{17b}$$

for $k = s, f, r$.

Expectation adaptation method is based on the assumption the expected values of the state variable vector between different time resolutions are the same. This adaptation method assumes that the ratio of the time steps (Δ) of the lower and higher resolutions is an integer. Specifically, if $\rho > 1, \rho \in \mathbb{Z}_+$, while if $\rho < 1, 1/\rho \in \mathbb{Z}_+$. Following Proposition 3 in Section Appendix A.1 the proposed mapping is

$$\boldsymbol{H}_{v,(T)} = \left(\boldsymbol{H}_{v,(S)}\right)^{\rho} \tag{18a}$$

$$\boldsymbol{H}_{k,(T)} = \begin{cases} (\sum_{j=1}^{\rho} \left(\boldsymbol{H}_{v,(S)}\right)^{\rho-j})\boldsymbol{H}_{k,(T)} & \rho \geq 1 \\ (\sum_{j=1}^{1/\rho} \left(\boldsymbol{H}_{v,(T)}\right)^{1/\rho-j})^{-1}\boldsymbol{H}_{k,(T)} & \rho < 1 \end{cases} \tag{18b}$$

for $k = s, f, r$.

Remark 2. In the Integral and Expectation adaptation methods, $\left(\boldsymbol{H}_{v,(S)}\right)^{\rho}$ represents a matrix raised to a possibly fractional power which can be computed using the Schur-Pade algorithm (Higham & Lin, 2011). In general, fractional powers of a real-valued matrix may be complex-valued. By restricting the parameters of the adLIF neuron (1) to the ranges $\alpha, \beta, a \in (0, 1)$, the possibly fractional power $\rho = 1/\bar{\rho}$ of the real matrix $\boldsymbol{H}_{v,(S)}$ used in our experiments remains real-valued (Higham & Lin, 2011).

4.4. A simple benchmark method: time-constant adaptation method

We now introduce a simple method that will serve as a benchmark for our proposed methods. This benchmark method is motivated by the explicit dependence of various neuron parameters on the sampling period in the popular LIF neuron and adaptive LIF neuron models, and has been utilized in the literature for the LIF neuron (He et al., 2020). Assume that the elements of the matrices $\boldsymbol{H}_{k,(S)}$ for $k = v, s, f, r$ belong to one of three groups:

1. in the form of $h_{(S)} = \exp^{-\Delta_S/\tau}$ where $\tau \in \mathbb{R}_+$;
2. linear transformation of $h_{(S)}$;
3. has no explicit dependency on Δ_S.

The time-constant adaptation methods obtain $\boldsymbol{H}iT$ for $k = v, s, f, r$, by replacing elements of $\boldsymbol{H}_{k,(S)}$ belonging to the group i,ii,iii by:

1. $h_{(T)} = \left(h_{(S)}\right)^{\rho}$
2. the same linear transformation of $h_{(T)}$
3. rest of the terms are unchanged

To illustrate the method, we now look at the popular adaptive LIF neuron from Section 3.1. In the adaptive LIF neuron, all of the three types of elements are present. Here, $\alpha_{(T)} = \alpha_{(S)}^{\rho}$ and $\beta_{(T)} = \beta_{(S)}^{\rho}$ due to their explicit exponential dependence on the step length T. The other parameters are either scaled accordingly due to the dependence with the already scaled parameters, for example $-(1-\alpha_{(T)})$, or kept the same (in the case of the parameters a and b).

Remark 3. The proposed methods of Section 4.3 do not require the parameters of the neuron model to explicitly depend on the time resolution and hence they are applicable to a wide range of neuron models. This is in contrast to the benchmark method of Section 4.4, where the method is best suited to the neuron models where all neuron parameters have explicit dependence on the time resolution.

4.5. Computational complexity

In this section, we discuss the computation complexity of the proposed methods based on the number of multiply-accumulate (MAC) operations needed.

4.5.1. Computational complexity of proposed adaptation methods mapping

In this section, we calculate the scaling of MAC operations for the proposed adaptation methods in terms of the number of neurons, neuron state dimensions, and target-to-source temporal resolution ratio ρ.

For the basic matrix operations, we use the following conventions: i) multiplying a $(n \times n)$ matrix and a $(n \times 1)$ vector requires $\propto n^2$ MACs; ii) inverting a $(n \times n)$ matrix requires $\propto n^3$ MACs; iii) finding $\rho > 1$ integer power of a $(n \times n)$ matrix, using repeated matrix power multiplication, requires $\propto (\rho - 1)n^3 \propto \rho n^3$ MACs; iv) finding $\rho < 1$ fractional power of $(n \times n)$ matrix requires $\propto (1/\rho)n^3$ MACs (Higham & Lin, 2011), (Higham & Al-Mohy, 2010, Section 4.2).

Using the general spiking neuron model in (2) and $\boldsymbol{H}_v \in \mathbb{R}^{n \times n}$ and $\boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r \in \mathbb{R}^{n \times 1}$, we obtain the following scaling of MAC operations per neuron for $\rho \geq 1$:

- Integral method in (16): $\rho\, n^3$ for $\boldsymbol{H}_{v,(T)}$ and n^3 for $\boldsymbol{H}_{k,(T)}$ for $k = s, f, r$.
- Euler method in (17): n^2 for $\boldsymbol{H}_{v,(T)}$ and n for $\boldsymbol{H}_{k,(T)}$ for $k = s, f, r$.
- Expectation method in (18): $\rho\, n^3$ for $\boldsymbol{H}_{v,(T)}$ and $\rho\, n^3$ for $k = s, f, r$.

For $\rho < 1$, ρ is replaced with $1/\rho$. We continue with the convention $\rho \geq 1$ in the below.

The above calculations are per neuron. For the whole network, these numbers are multiplied by the number of neurons, i.e., $\sum_{l=1}^{L} h_l$, where h_l is the number of neurons in the hidden layer l and L is the total number of hidden layers. Hence, assuming all hidden layers have h_l neurons, number of MAC operations scale with $\rho\, n^3 h_l$ for Integral method, $n^2 h_l$ for Euler method and $\rho n^3 h_l$ for Expectation method.

We note that only the ratio of the target and source sequence lengths, i.e. ρ (in (6)), affect the computational cost. In other words, the length of the source or target sequence does not directly impact the computational complexity of our adaptation methods.

4.5.2. Computational complexity of inference

The adaptation of the model is performed once. After the adaptation, inference proceeds normally. Our methods operate by replacing existing neuron parameters with the adapted values, without introducing any additional parameters. Thus, the total parameter count of the network remains unchanged, and the number of operations per time step during inference is identical to that of the original, non-adapted model.

We now provide an overview of the computational complexity cost during inference, i.e., forward pass. The dominant MAC cost during inference comes from the weight matrix multiplications which scales with $\sum_{l=1}^{L} h_{l-1} h_l$ for a feedforward network and $\sum_{l=1}^{L} h_{l-1} h_l + h_l^2$ for a recurrent network. This is due to the fact that in a typical SNN, the state dimension of neurons n, is much smaller than the number of neurons in the layer h_l, i.e. $h_l \gg n$, hence the typical computational cost due to the update of dynamics is insignificant. Hence, the MAC per time step during inference grows with h_l^2. For a sequence with T number of time steps, the inference cost then becomes $T h_l^2$. Hence, the computational cost of inference, under both the adapted and non-adapted models, scales with $T h_l^2$.

4.5.3. Comparison between cost of adaptation and re-training

In this article, we focus on a zero-shot setup where the target data is unavailable and re-training cannot be performed. For comparison purposes, here we discuss the computational cost of re-training, and compare it with the cost of our adaptation methods.

The computational complexity of re-training depends on the computational cost of backward pass. A common approximation is that the computational backward pass requires about twice the cost of the forward pass (Wiedemann et al., 2020) (Narayanan et al., 2021, Fig. 3). Thus, for sequence of length T, since inference complexity scales with $\propto T h_l^2$ (see Section 4.5.2), training complexity scales with $3T h_l^2$ including both inference and backward pass. Moreover, since training is typically performed over multiple data samples, the scaling of cost of re-training becomes $3BT h_l^2$ per training step where B is the number of samples.

We now compare the cost of adaptation, which scales with $\rho n^3 h_l$ at worst, with the cost of re-training, which scales with $3BT h_l^2$ for a single epoch. We note that in SNNs typically $h_l \gg n$, and $B, T \gg 1$ so that it is typically the case that the scaling satisfies $3BT h_l^2 \gg \rho n^3 h_l$. This represents a substantial difference in scaling of the complexity for SNNs with large h_l, highlighting the efficiency fo scaling of our proposed methods of adaptation over re-training. Moreover, in contrast to re-training, the computational cost of our proposed adaptation methods are independent of the sequence length. As a result, our approach scales much better than full re-training when dealing with higher-resolution data or long sequences.

To illustrate the possible difference in the scaling of computational cost, we now provide typical values for n, h_l, T and B. For both SHD and MSWC datasets, this article and the prior works (Bittar & Garner, 2022; Yik, 2025) use RadLIF neurons with $n = 2$ and hidden layers with $h_l = 1024$ neurons. In our setting, sequence length in its highest resolutions for SHD dataset is $T = 100$, and for MSWC, it is $T = 201$. Under these settings, for our zero-shot adaptation, we have $\rho n^3 h_l = \rho \times 2^3 \times 1024$, whereas the scaling of re-training per epoch per sequence is $T h_l^2 = 100 \times 1024^2$ for SHD dataset. Moreover, training a single epoch over the full training dataset would multiply the cost of training by $B = 8332$ for SHD, and $B = 5 \times 10^4$ for MSWC dataset.

The developments in this section provide approximate proportional scalings of the computational costs. Their reflection in practice depends on variable ranges, hardware specifics and implementation

details. In Section 5.6, we provide empirical comparisons based on actual wall-clock times.

5. Results

In Section 5.1, we define our experimental setup. Section 5.2 illustrates the performance of the proposed methods on a single neuron using a synthetic dataset. Section 5.3 presents the baseline performances in various temporal resolutions with event-based datasets. Sections 5.4 and 5.5 present the performance of the proposed temporal domain adaptation methods on these datasets. This motivates us to illustrate in Section 5.6 that the proposed methods can be used to support time-efficient training.

The research questions targeted by these experiments are the following:

- Can the proposed methods for zero-shot adaptation without retraining achieve satisfactory accuracy levels on varying temporal resolution levels?
- How do the proposed methods perform relative to the benchmark methods?
- Can we train efficiently at low temporal resolution and infer at high resolution with minimal performance loss using the proposed methods?

5.1. Preliminaries

5.1.1. Datasets

We use the following event-based datasets:

The Spiking Heidelberg Digits (SHD) dataset (Cramer et al., 2022) is an audio dataset consisting of non-signed spikes generated using the mathematical artificial cochlea model called Lauscher. The dataset consists of 20 classes of spoken digits from 0 to 9 in both German and English language with total of 8156 train samples, 2264 test samples. Each sample consists of 700 input channels each with 100 time steps, where each time step represents summed spikes over the time window of length 1ms. The benchmark for SHD dataset is 95.1 % test classification accuracy (Hammouamri et al., 2024) where SNN with learnable delays are used. The closest setup to our work achieves 94.6 % accuracy (Bittar & Garner, 2022), using the same SNN architecture and recurrent adaptive LIF neuron. However, their best model was selected based on the test dataset rather than a separate validation dataset, making the benchmark overly optimistic.

The Multilingual Spoken Word Corpus (MSWC) dataset (Mazumder et al., 2024) is a large and growing audio dataset. In this paper we use a subset of samples and encoding as proposed in the NeuroBench initiative (Yik, 2025). The subset consists of 100 classes, 20 classes per 5 languages. We use 600 train, and 100 test samples per class. Each audio sample is converted to signed spikes using the Speech2Spikes (S2S) (Stewart et al., 2023) preprocessing algorithm resulting in 20 input channels and 201 time steps. The current baseline for this subset of MSWC is 93.48 % classification accuracy (Yik, 2025).

The NMNIST dataset (Orchard et al., 2015) is a neuromorphic version of the MNIST vision dataset. It consists of 10 classes representing the digits 0 to 9 in an event-based manner. Samples are obtained by recording the static MNIST images shown on a monitor using a mobile event based image sensor (Orchard et al., 2015). The dataset consists of 60000 train and 10000 test samples. We use the Tonic library (Lenz et al., 2021) for converting each sample into $(34 \times 34 \times 2, N)$ spike pattern, where the first dimension represents spatial information such as channel and height and widths of the frame, while the second dimension is the number of time steps. We take $N \approx 300$ where the exact number differs due to the Tonic denoise filter. The state of the art for NMNIST dataset is 99.6 % accuracy using convolutional SNNs (Samadzadeh et al., 2023).

5.1.2. Overview of experimental procedure

We define the data in the above datasets as having the fine temporal resolution, and refer them as the scenario of bin size 1, i.e. $b = 1$. We create data for coarser resolutions ($b = 2, 3, 4, 10$) using the sum-binning procedure described in Section 5.1.4. We have two sets of experiments: Coarse-to-Fine and Fine-to-Coarse deployment. In the Coarse-to-Fine experiments, the models are trained on a simulated coarse-resolution version of the train dataset, and evaluated on the test dataset with the unaltered fine resolution. In the Fine-to-Coarse experiments, the models are trained on the unaltered fine-resolution dataset, and evaluated on the simulated coarser-resolution version of the test dataset.

All the values reported in this paper are average performance of 10 different initializations of the models.

5.1.3. Model architecture and training

For all experiments we use a SNN with adaptive LIF (adLIF) neuron dynamics (Bittar & Garner, 2022) with 2 hidden layer of 1024 neurons. In the results, this setup is referred to as adLIF for the model without recurrent connections and RadLIF for the model with recurrent connections. Each model is trained for 50 iterations, and we report the test data accuracy at the final (50-th) iteration. Models are trained using BPTT with a box surrogate gradient (Neftci et al., 2019) as implemented by Bittar & Garner (2022). Models are trained on Nvidia A100 GPU with 64GB system memory, 40GB VRAM and 16 CPU cores. See Section Appendix A.4 for further details on hyperparameters.

5.1.4. Data with coarse temporal resolution

Given a sequence, we obtain a coarser time resolution version of it using sum-binning with a non-overlapping moving window along its time dimension. For example, given a sequence of length 4, say (x, y, z, w) where $x, y, z, w \in \mathbb{Z}$, by applying non-binary sum binning with bin size $b = 2$, we obtain $(x + y, z + w)$. Note that increasing the bin size smooths over temporal features, leading to a loss of temporal detail and reduced information in the data.

5.1.5. Batchnorm-scaling

One dimensional Batch Normalization layers (Ioffe & Szegedy, 2015) are used before each spiking layer in the SNNs. These layers are adapted due the change of temporal resolution between target and source data, see Section Appendix A.2 for details.

5.1.6. Benchmark methods for temporal resolution domain adaptation

In both Fine-to-Coarse and Coarse-to-Fine experiments, two benchmarks are used: (i) No-adaptation of the model; (ii) Time-constants adaptation benchmark of Section 4.4.

5.2. Illustrative example: neuron dynamics under proposed methods

We now investigate the performance of our proposed temporal domain adaptation methods at a single neuron level. We use the adLIF neuron as in (1). Specifically, as we have a single neuron without recurrent connections to other neurons, (1a) becomes

$$u[t+1] = \alpha(u[t] - \theta s_{out}[t]) + (1-\alpha)i[t] - (1-\alpha)u[t], \tag{19}$$

while (1b) and (1c) remain the same. As input to the neuron, we use an excitation signal commonly used in signal processing and system identification for its ability to excite a wide range of system dynamics, i.e., we use the input

$$i[t] = \sum_{k=1}^{K} A_k \sin[\omega_k t + \pi/\phi_k] \tag{20}$$

which represents a linear combination of K sin waves, each with a different amplitude A_k, frequency ω_k, and phase shift π/ϕ_k. The maximum number of time steps is set to $T = 100$ i.e. $t = 0, 1, \ldots, 100$. We then also create $i_2[t]$, a sum binned version of $i[t]$ with $b = 2$ and $T = 50$. The general objective is to explore to which extent the dynamic behavior of the

Table 2
Qualitative match of the voltage dynamics for various adaptation methods. Each value is average of 1000 different combination of adLIF neuron and input sequence. b_S and b_T stand for source and target bin size.

(a) $b_S = 1$ to $b_T = 2$		
Model Adapt.	Q_1	Q_2
None	0.66 ± 0.16	0.84 ± 0.08
Integral	**0.92 ± 0.10**	**0.97 ± 0.04**
Expectation	0.91 ± 0.11	0.96 ± 0.04
Euler	0.87 ± 0.11	0.95 ± 0.03
Time-const.	0.77 ± 0.11	0.95 ± 0.03
(b) $b_S = 2$ to $b_T = 1$		
Model Adapt.	Q_1	Q_2
None	0.57 ± 0.25	0.84 ± 0.08
Integral	**0.97 ± 0.05**	**0.99 ± 0.01**
Expectation	**0.97 ± 0.03**	**0.99 ± 0.01**
Euler	0.96 ± 0.06	**0.99 ± 0.02**
Time-const.	0.84 ± 0.07	0.93 ± 0.03

neuron can be preserved despite the changes in temporal resolution of the input data.

In the Fine-to-Coarse scenario, i.e. source bin $b_S = 1$ to target bin $b_T = 2$, first the high resolution $i[t]$ is given as input to the adLIF neuron and $uNoArg[t]$ is recorded. We refer to this as reference dynamics. We then apply different model adaptations for the parameters of the adLIF neuron and record $uNoArg[t]$ when sending $i_2[t]$ as input to the neuron. In the Coarse-to-Fine scenario, i.e. $b_S = 2$ to $b_T = 1$, $i_2[t]$ constitutes the reference dynamics, and the performance of different model adaptations under the high resolution input $i[t]$ is evaluated.

To quantitatively inspect the mismatch between the reference and the adapted dynamics, we use the quality functions Q_1 and Q_2, which are associated with the normalized relative square error, and the correlation coefficient, respectively. For both Q_1 and Q_2, having a value close to 1 indicates a better reconstruction compared to a value close to 0. See Section Appendix A.3 for exact definitions.

The input $i[t]$ is defined as in (20) with $K = 3$, $A_k \in [0.1, 0.2]$, $\omega_k \in [1, 10]$, and $\phi_k \in [1, 20]$, and adLIF neurons with $\alpha \in [0.6, 0.98]$, $\beta \in [0.6, 0.98]$, $a \in [0.2, 0.5]$, and $b \in [0.2, 0.5]$. We simulate 1000 random pairs of input sequences and adLIF neurons from these intervals and calculate the average Q_1 and Q_2 over all instances. The resulting values are presented in Table 2. Two illustrative cases with their corresponding membrane potential $uNoArg[t]$ are provided in Fig. 3. On average, the results in Table 2 indicate a better overall match under the three proposed adaptation methods (Integral, Expectation and Euler) compared to both the Time-constant adaptation baseline and the No adaptation case. Time-constant adaptation method performs relatively poor compared to the proposed methods according to Q_1, and relatively close to the these methods according to Q_2. Fig. 3 illustrate that although the integral adaptation method may match the reference dynamics better than no adaptation, noticeable deviations compared to the original signal may also occur.

5.3. Baseline performance for different temporal resolutions

In Table 3 we present the performance of SNN models trained and tested on the same time resolution, i.e., the source and target bin sizes are equal, $b_S = b_T$. We refer to these results as baseline performance since they show the accuracy levels and computational time that would be achieved if the models were trained with source data whose temporal resolution is the same as in the target data.

Note that lowering the time resolution might result in drop in accuracy due to potentially removing finer details in the time sequence by eliminating or deforming important features needed for the classification task. However, higher time resolutions require more backpropagation steps during training, which may negatively affect performance due to the limitations of surrogate gradients in BPTT for SNNs. This phenomenon can also be seen in [He et al. (2020), Table VI, VII, VIII]. Table 3 illustrates both of these possibilities: for the SHD dataset (Table 3(a)), accuracy decreases monotonically with lower resolution, while for the MSWC dataset (Table 3(b)), accuracy initially increases slightly before declining.

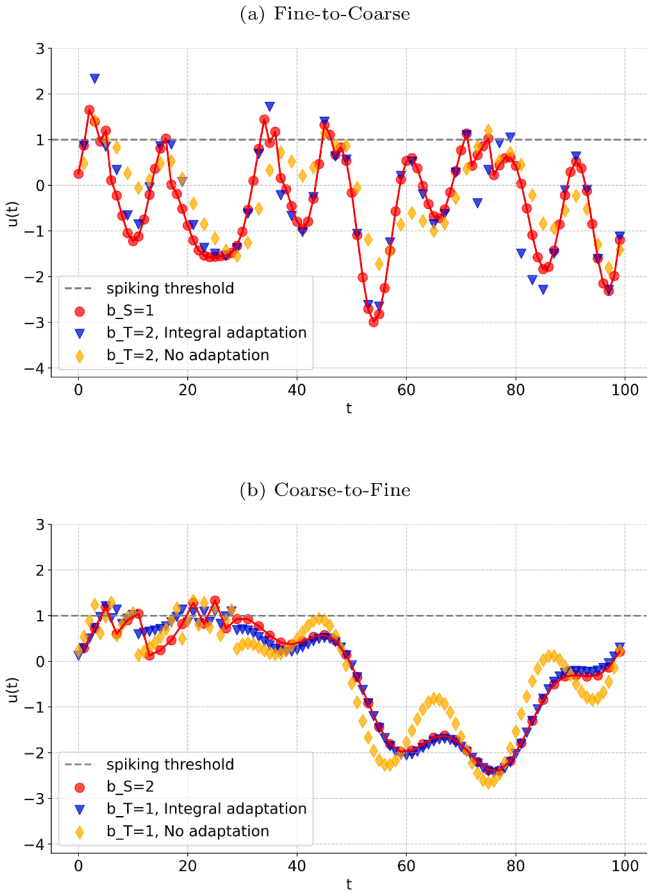


Fig. 3. Membrane potential dynamics of a single adLIF neuron over time for different adaptation methods and time resolutions where b_S (red circular) is the reference dynamics.

Compared to the SHD and MSWC datasets, NMNIST shows a smaller decline in accuracy as the bin size increases. This is consistent with the static nature of the NMNIST data, in contrast to the dynamic audio data, where temporal features play a key role in encoding information.

Although various anti-aliasing filters can be used to achieve lower temporal resolution, sum-binning offers advantages in terms of simplicity, real-time application, and preserving the event-based nature of the data. As shown in Table 3, the accuracy remains largely unaffected for bin sizes of 2 to 4, indicating minimal impact from aliasing noise or the exclusion of high-frequency content. However, a significant performance drop is observed with a bin size of 10, suggesting that important temporal content is lost under $b = 10$.

5.4. Coarse-to-fine temporal resolution domain adaptation

In Table 4 we present the performance of the proposed adaptation methods under Coarse-to-Fine deployment. In all cases and datasets inspected, the proposed adaptation methods significantly outperform no-adaptation scenario and the benchmark Time-constant adaptation method. As discussed in Section Appendix A.1, Expectation and Integral adaptation methods provide identical scaling under certain common settings, hence their adaptation performance is almost identical. In most cases, the accuracy provided by Euler adaptation method is

Table 3
Baseline results for training SNN models using various bin sizes, where the source and target bin sizes are equal $b_S = b_T$. Two different neuron models are used: adLIF and RadLIF. Each value is average over 10 trials. The benchmark results from the literature presented in Section 5.1 are obtained using the highest temporal resolution data i.e. $b = 1$.

Bin (b)	Steps (N)	Δ (ms)	adLIF Train (min)	adLIF Accuracy	RadLIF Train (min)	RadLIF Accuracy
(a) Results for SHD dataset.						
1	100	1	6.3	91.5 ± 0.6 %	7.7	91.8 ± 0.7 %
2	50	2	2.7	90.3 ± 0.6 %	3.6	91.0 ± 1.0 %
3	33	3	1.8	89.8 ± 0.8 %	2.7	90.7 ± 0.7 %
4	25	4	1.6	89.1 ± 0.8 %	2.3	88.9 ± 0.9 %
10	10	10	1.1	81.8 ± 1.1 %	1.7	79.2 ± 1.5 %
100	1	100	0.7	50.2 ± 1.8 %	1.3	6.3 ± 2.8 %
(b) Results for MSWC dataset.						
1	201	5	106.2	92.4 ± 0.3 %	117.0	95.7 ± 0.1 %
2	100	10	36.9	93.5 ± 0.2 %	41.7	95.9 ± 0.1 %
3	67	15	21.8	94.1 ± 0.2 %	24.7	95.7 ± 0.2 %
4	50	20	15.1	93.8 ± 0.2 %	17.5	95.6 ± 0.2 %
10	20	50	6.3	90.7 ± 0.3 %	7.3	92.8 ± 0.4 %
201	1	1000	2.3	1.0 ± 0.1 %	2.6	1.0 ± 0.1 %
(c) Results for NMNIST dataset.						
1	300	1	433.1	98.6 ± 0.1 %	487.9	98.5 ± 0.1 %
2	150	2	223.3	98.7 ± 0.1 %	245.4	98.5 ± 0.1 %
3	100	3	150.9	98.7 ± 0.1 %	178.1	98.5 ± 0.1 %
4	75	4	112.6	98.6 ± 0.0 %	143.4	98.5 ± 0.1 %
10	30	10	104.4	98.7 ± 0.1 %	87.4	98.3 ± 0.1 %
300	1	300	84.4	92.5 ± 0.5 %	84.9	11.3 ± 0.0 %

lower compared to Expectation and Euler, but still significantly higher than no-adaptation and Time-constant benchmark methods. As shown in Section Appendix A.1, the Euler method is based on the first-order approximation of the derivative, and a simpler approach compared to the Expectation and Integral method. The lower accuracy of Euler method compared to these two methods is consistent with higher information loss due to its simpler approach.

For all model adaptation methods, in general, the performance declines as the difference between the source and target temporal resolutions increases. This is consistent with the growing mismatch between the source and target feature space, which hinders the model's ability to capture essential patterns, either because these features are not present or are inadequately represented in the source data.

We now compare the adLIF and RadLIF columns of Table 4 and the corresponding baseline performance in Table 3. We observe that in the majority of the cases, the temporal domain adaptation accuracy in the recurrent SNNs, i.e. RadLIF columns, has a larger drop of performance compared to the non-recurrent SNNs, i.e. adLIF columns. This difference might be due to the fact that we have associated the recurrent connection in the neuron dynamics in the recurrent SNNs with a linear external input in SSMs, see Section 4.2. When there is no recurrent connection in SNN (adLIF) this association is not required, and hence the correspondence between SNNs and linear SSMs provides a (still approximate but) closer match.

We now compare the model adaptation performance under NMNIST with other datasets. The accuracy drop, i.e. the gap between the accuracy in Table 4 and the baseline accuracy in Table 3, is typically smaller for NMNIST than that for SHD and MSWC. For example, for the scenario of bin size $b_S = 10$ to bin size $b_T = 1$ with the best performing adaptation method under adLIF, the accuracy drop is 21 %, 5.6 %, 0.9 %, for SHD, MSWC and NMNIST, respectively. Possible reason for this behavior is again the static nature of the NMNIST data, where the time dimension does not encode any inherent temporal information unlike in the audio datasets where the time behavior is important for distinguishing between the different classes. However, including the NMNIST

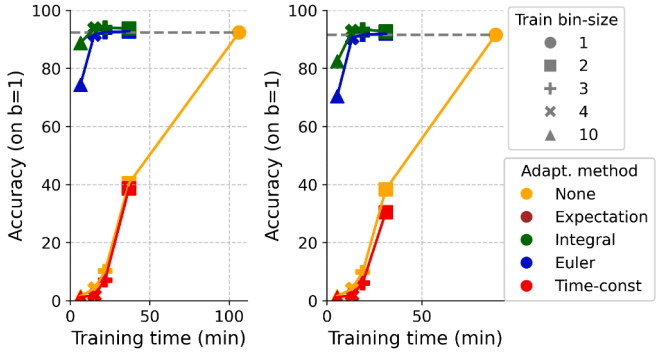


Fig. 4. Both plots show training + adaptation (wall-clock) time vs accuracy for the MSWC dataset and adLIF neuron, where the left plot uses fixed 50 epochs pre-training (results in Table 4(b)), while the right plot uses early stopping pre-training. The gray dashed line represent the baseline performance of pre-training the model on the target resolution. Note that lines for the Integral and Expectation adaptation methods are overlapping.

dataset in the evaluation has shown that our methods can be applied and expected to perform in different mediums of data origin.

5.5. Fine-to-coarse temporal resolution domain adaptation

In Table 5, we present the Fine-to-Coarse deployment performance of the proposed parameters adaptation methods. Comparing the results with the baseline performance in Table 3, we observe that there is a relatively large gap in performance for all scenarios with SHD and MSWC, which suggests that the fine-to-coarse deployment with spiking neurons is challenging. Now we compare our proposed method with the no-adaptation baseline in Table 5. The proposed Integral and Expectation methods consistently outperform the no-adaptation baseline or demonstrate statistically equivalent performance considering the standard deviations. In contrast, the average performance of the Euler method is generally inferior to that of the no-adaptation baseline, although in some cases it remains within the standard deviation range of the baseline. Now we compare our proposed method with the time-constant adaptation method. Under adLIF, time-constant adaptation method provides the best performance in several cases whereas under RadLIF the proposed methods perform better than time-constant adaptation method in all scenarios, indicating that the proposed methods can address the challenge of scaling the recurrent external feedback in a better manner.

5.6. Time efficient training

We now investigate the trade-offs between accuracy and training time for pre-training the model on various temporal resolutions, i.e. varying b_S, while evaluating its performance on $b_T = 1$.

In the previous sections, the pre-trained models were obtained using a fixed number of training iterations (50) over the training dataset. To reveal the accuracy-complexity trade-offs better, here we introduce an additional pre-training setup that uses a validation set. (If the dataset does not have a pre-defined validation data, we extract a validation set from the training data, matching the size of the test set.) We implement early stopping training, where training continues until convergence based on validation data loss. Specifically, training is allowed up to 100 iterations, with early termination if the reduction in validation loss is less than 10^{-5} for 10 consecutive iterations. The best-performing model in the validation set is then saved, and its accuracy on the test dataset is reported.

We focus on the MSWC dataset and adLIF neuron for illustration. In Fig. 4, the accuracy on bin size $b_T = 1$ is plotted against the total training time, where the total training time is the sum of the training time at source resolution b_s (Step 1 of Algorithm 1) and the adaptation time if it is applied (Step 2 of Algorithm 1).

Table 4
Performance under various domain adaptation methods on target data with fine temporal resolution ($b_T = 1$), where the models were trained on source data with coarser temporal resolution ($b_S = 2, 3, 4, 10$). Each value in the table is average of 10 trials.

(a) Results using SHD Dataset. Baseline accuracy ($b_S = b_T = 1$) from Table 3(a) is 91.4 % for adLIF and 91.6 % for RadLIF.								
	$b_S = 2$ to $b_T = 1$		$b_S = 3$ to $b_T = 1$		$b_S = 4$ to $b_T = 1$		$b_S = 10$ to $b_T = 1$	
Model Adapt.	adLIF	RadLIF	adLIF	RadLIF	adLIF	RadLIF	adLIF	RadLIF
None	58.1 ± 1.9 %	31.8 ± 4.6 %	30.3 ± 2.1 %	5.3 ± 0.4 %	20.5 ± 1.7 %	6.1 ± 1.5 %	9.6 ± 1.6 %	6.7 ± 2.1 %
Expectation	**89.5 ± 0.6** %	**87.7 ± 2.3** %	**87.4 ± 0.4** %	**81.6 ± 1.9** %	**84.3 ± 0.6**%	**72.8 ± 2.6** %	**70.4 ± 1.6** %	**45.4 ± 3.8** %
Integral	**89.5 ± 0.6** %	**87.7 ± 2.3** %	**87.4 ± 0.4** %	**81.6 ± 1.9** %	**84.3 ± 0.6** %	**72.8 ± 2.6** %	**70.4 ± 1.7** %	**45.4 ± 3.9** %
Euler	84.6 ± 0.6 %	76.8 ± 7.0 %	77.8 ± 1.2 %	57.5 ± 5.7 %	71.3 ± 2.1 %	50.5 ± 7.4 %	49.7 ± 1.2 %	45.3 ± 4.4 %
Time-const.	53.0 ± 3.2 %	7.5 ± 2.9 %	19.9 ± 3.9 %	5.6 ± 1.5 %	14.3 ± 4.1 %	5.0 ± 0.2 %	5.0 ± 0.0 %	5.0 ± 0.1 %
(b) Results using MSWC Dataset. Baseline accuracy ($b_S = b_T = 1$) from Table 3(b) is 92.4 % for adLIF and 95.7 % for RadLIF.								
None	40.5 ± 1.9 %	22.2 ± 2.2 %	10.2 ± 0.5 %	3.4 ± 0.6 %	4.2 ± 0.2 %	1.7 ± 0.3 %	1.6 ± 0.2 %	1.3 ± 0.2 %
Expectation	**93.6 ± 0.3** %	94.2 ± 0.3 %	**94.0 ± 0.2** %	**92.9 ± 0.4** %	**93.8 ± 0.1** %	**92.1 ± 0.9** %	**86.8 ± 1.0** %	**61.8 ± 9.1** %
Integral	**93.6 ± 0.3** %	94.2 ± 0.3 %	**94.0 ± 0.2** %	**92.9 ± 0.4** %	**93.8 ± 0.1** %	**92.1 ± 0.9** %	**86.8 ± 1.0** %	**61.8 ± 9.2** %
Euler	92.8 ± 0.2 %	**94.9± 0.2**%	92.6 ± 0.3 %	92.3 ± 0.6 %	91.7 ± 0.2 %	87.7 ± 1.0 %	74.4 ± 1.1 %	23.9 ± 3.8 %
Time-const.	38.8 ± 2.8 %	1.1 ± 0.2 %	7.1 ± 1.5 %	1.0 ± 0.1 %	1.9 ± 0.8 %	1.1 ± 0.2 %	1.2 ± 0.2 %	1.0 ± 0.1 %
(c) Results using NMNIST Dataset. Baseline accuracy ($b_S = b_T = 1$) from Table 3(c) is 98.6 % for adLIF and 98.5 % for RadLIF.								
None	95.4 ± 0.7 %	97.2 ± 0.4 %	83.9 ± 2.3 %	93.3 ± 2.5 %	72.8 ± 3.8 %	86.3 ± 6.2 %	47.9 ± 3.9 %	18.5 ± 10.7 %
Expectation	**98.5 ± 0.1** %	**98.5 ± 0.1** %	98.1 ± 0.2 %	**98.4 ± 0.2** %	94.9 ± 0.9 %	**98.3 ± 0.2** %	76.9 ± 5.2 %	**96.0 ± 1.2** %
Integral	**98.5 ± 0.1** %	**98.5 ± 0.1** %	98.1 ± 0.2 %	**98.4 ± 0.2** %	94.9 ± 0.9 %	**98.3 ± 0.2** %	76.9 ± 5.2 %	**96.0 ± 1.2** %
Euler	98.4 ± 0.1 %	98.3 ± 0.1 %	**98.5 ± 0.1** %	98.2 ± 0.2 %	**98.5 ± 0.1** %	98.2 ± 0.2 %	**97.7 ± 0.2** %	94.6 ± 2.4 %
Time-const.	97.2 ± 0.3 %	73.8 ± 15.5 %	89.4 ± 5.3 %	23.6 ± 17.8 %	46.0 ± 13.0 %	13.4 ± 5.1 %	20.8 ± 6.3 %	10.5 ± 0.9 %

Table 5
Performance under various domain adaptation methods on target data with coarser temporal resolution ($b_T = 2, 3, 4, 10$), where the models were trained on source data with fine time resolution ($b_S = 1$). Each value in the table is average of 10 trials.

(a) Results using SHD dataset.								
	$b_S = 1$ to $b_T = 2$		$b_S = 1$ to $b_T = 3$		$b_S = 1$ to $b_T = 4$		$b_S = 1$ to $b_T = 10$	
Model Adapt.	adLIF	RadLIF	adLIF	RadLIF	adLIF	RadLIF	adLIF	RadLIF
None	56.9 ± 1.6 %	45.4 ± 2.5 %	26.7 ± 2.2 %	21.5 ± 1.4 %	18.4 ± 1.9 %	13.0 ± 1.7 %	10.7 ± 1.5 %	8.2 ± 2.0 %
Expectation	75.1 ± 2.2 %	**76.0 ± 2.6** %	52.0 ± 2.8 %	**45.6 ± 4.0** %	34.3 ± 2.5 %	**29.3 ± 3.9** %	14.4 ± 1.8 %	**9.0 ± 1.1** %
Integral	75.1 ± 2.2 %	**76.0 ± 2.6** %	52.0 ± 2.8 %	**45.6 ± 4.0** %	34.3 ± 2.5 %	**29.3 ± 3.9** %	14.4 ± 1.8 %	**9.0 ± 1.1** %
Euler	55.5 ± 4.7 %	50.0 ± 9.3 %	7.1 ± 1.0 %	7.3 ± 1.7 %	5.8 ± 0.8 %	5.7 ± 0.8 %	6.2 ± 1.0 %	4.7 ± 1.1 %
Time-const.	**80.1 ± 2.2** %	54.4 ± 11.2 %	**55.0 ± 4.3** %	19.5 ± 8.2 %	**38.4 ± 3.7** %	11.2 ± 4.4 %	**17.3 ± 2.5** %	6.8 ± 1.4 %
(b) Results using MSWC dataset.								
None	54.1 ± 2.6 %	32.0 ± 3.3 %	13.4 ± 1.4 %	2.5 ± 0.5 %	4.6 ± 0.4 %	1.4 ± 0.2 %	1.4 ± 0.2 %	1.1 ± 0.1 %
Expectation	**59.4 ± 12.5** %	42.7 ± 5.0 %	**25.9 ± 8.7** %	4.0 ± 1.1 %	**13.1 ± 3.3** %	2.0 ± 0.4 %	**2.7 ± 0.3** %	1.1 ± 0.2 %
Integral	**59.4 ± 12.5** %	42.7 ± 5.0 %	**25.9 ± 8.7** %	4.0 ± 1.1 %	**13.1 ± 3.3** %	2.0 ± 0.4 %	**2.7 ± 0.3** %	1.1 ± 0.2 %
Euler	45.6 ± 5.1 %	**49.2 ± 4.4** %	6.9 ± 2.2 %	**5.0 ± 1.2** %	2.5 ± 0.8 %	**2.4 ± 0.5** %	1.4 ± 0.3 %	**1.2 ± 0.1** %
Time-const.	55.5 ± 15.3 %	6.8 ± 1.3 %	19.9 ± 9.2 %	1.5 ± 0.3 %	7.9 ± 3.7 %	1.2 ± 0.2 %	2.0 ± 0.5 %	1.0 ± 0.2 %
(c) Results using NMNIST dataset.								
None	96.4 ± 0.7 %	97.5 ± 0.3 %	92.7 ± 1.8 %	**96.3 ± 0.6** %	88.3 ± 2.8 %	**94.8 ± 1.2** %	60.1 ± 5.8 %	74.2 ± 6.3 %
Expectation	**98.3 ± 0.2** %	**98.0 ± 0.3** %	97.2 ± 0.5 %	**96.3 ± 1.1** %	95.3 ± 1.1 %	94.3 ± 2.0 %	83.7 ± 4.0 %	**89.9 ± 3.5** %
Integral	**98.3 ± 0.2** %	**98.0 ± 0.3** %	97.2 ± 0.5 %	**96.3 ± 1.1** %	95.3 ± 1.1 %	94.3 ± 2.0 %	83.7 ± 4.0 %	**89.9 ± 3.5** %
Euler	88.9 ± 3.6 %	75.6 ± 12.6 %	88.6 ± 4.2 %	75.3 ± 11.3 %	87.6 ± 3.7 %	78.8 ± 10.3 %	73.5 ± 7.3 %	80.4 ± 9.3 %
Time-const.	98.2 ± 0.2 %	81.2 ± 14.4 %	**97.9 ± 0.2** %	71.5 ± 17.0 %	**97.7 ± 0.3** %	70.6 ± 16.0 %	**96.5 ± 0.7** %	77.3 ± 8.6 %

We observe that for both figures, i.e. the fixed epochs pre-training and the early-stopping pre-training, the general characteristics of the curves are the same. In particular, they show that significant gains can be obtained by pre-training lower resolutions. For instance, consider the results with the early stopping procedure. Instead of training a model on data with bin size 1 until convergence, an alternative approach is to train on bin size $b_S = 4$ and apply the Integral adaptation method to achieve a model with high-accuracy on data with bin size $b_T = 1$. In particular, we obtain an accuracy of 93.8 % instead of the baseline performance of 92.4 % while training $106/15 \approx 7$ times faster.

We note that the adaptation procedure (Step 2 of Algorithm 1) takes a small fraction of the total time. For example, for the setting in Fig. 4, it takes a maximum of 0.012 minute to adapt the parameters of all neurons in the network using any of our proposed methods, whereas training for a single epoch with bin size 1 requires $\approx 106/50 = 2.12$ minutes. Hence, consistent with the discussions in Section 4.5.3, adaptation is significantly more efficient than re-training.

These results suggest that training on low temporal resolution datasets and adapting models using the proposed methods may significantly reduce training time with minimal to no impact on accuracy when source and target resolutions are close, with no re-training on the target dataset.

6. Ablation study

In Section 6.1 we investigate the effect of adapting the normalization layer to the change of temporal resolution in the data. In Section 6.2 we investigate different types of adaptation, where instead of adapting the models as in our proposed methods, we adapt the input to the model. In Section 6.3, we investigate the performance when the coarse resolution sequences are obtained using different methods. We evaluate the impact of spike sparsity on the performance of our methods in Section 6.4. We discuss an alternative method for linearization of the neuron dynamics in Section 6.5.

Table 6
Comparison between the performance contribution of model adaptation and normalization layer adaptation, for Coarse-to-Fine and Fine-to-Coarse scenario with MSWC dataset.

Model Adapt.	Stats used for norm.	Norm. Adapt.	$b_S = 1$ to $b_T = 2$ adLIF	$b_S = 1$ to $b_T = 2$ RadLIF	$b_S = 2$ to $b_T = 1$ adLIF	$b_S = 2$ to $b_T = 1$ RadLIF
Integral	Source	None	37.7 ± 1.7	51.4 ± 3.4	25.5 ± 2.5	20.2 ± 7.1
Integral	Source	Adapt	75.4 ± 2.3	75.8 ± 2.8	89.5 ± 0.6	87.6 ± 2.4
Integral	Target	N/A	83.2 ± 1.5	86.5 ± 1.0	91.5 ± 0.6	90.2 ± 1.7
None	Source	None	29.2 ± 2.5	30.0 ± 2.7	21.2 ± 2.4	8.2 ± 3.1
None	Source	Adapt	57.1 ± 1.7	45.3 ± 2.5	58.2 ± 1.7	31.6 ± 4.6
None	Target	N/A	71.2 ± 1.7	60.0 ± 2.1	66.7 ± 1.3	45.8 ± 2.4

6.1. Normalization layer adaptation

In the previous section, we utilize model adaptation of Section 4.3 and normalization layer adaptation, of Section Appendix A.2 together. We now evaluate the effect of these components separately.

Table 6 provides the results for Coarse-to-Fine ($b_S = 2$ to $b_T = 1$) and Fine-to-Coarse ($b_S = 1$ to $b_T = 2$) deployment for SHD dataset. "Stats used for normalization" indicates the data statistics used in the normalization layers of the model. Recall that results presented previously correspond to the row "Integral-Source-Adapt". Note that the label "target" represents the case where statistics of the target dataset is used for normalization. This represents a potentially impractical but useful high-performance baseline since target data statistics provide the ideal normalization for the target dataset but cannot be always calculated.

We now focus on the cases where target data is used for normalization. We observe that when the Integral model adaptation method is applied, a significant performance improvement compared to the case of no model adaptation is obtained, e.g. an improvement from 66.7 % to 91.5 % under $b_S = 2$ to $b_T = 1$ with adLIF. This indicates that accurate normalization alone cannot fully address the feature shift between source and target data.

Furthermore, examining the cases where the source data statistics are used with no normalization adaptation, for example in the case of $b_S = 1$ to $b_T = 2$ and RadLIF, we again observe a significant performance improvement from 30.0 % to 51.4 % when the Integral model adaptation method is applied compared to the no model adaptation. This indicates that even in the case of no normalization the proposed model adaptation method offer significant improvement.

These results suggest that, in practical applications where target data statistics are inaccessible, the combination of our model adaptation methods and proposed normalization adaptation should be preferred. This combination is used in the previous sections, such as in Table 4 and Table 5.

6.2. Input adaptation

Our proposed adaptation methods adapt the model to account for the change in the temporal resolution of the source and target data. In this section, we look into the scenario where the model is kept fixed but instead we transform the target data to mimic the temporal resolution of the source data. In this approach, the length of the target data is adjusted at deployment time to match the length of the source data.

The normalization layers adaptation for the input adaptation method presented in this section are stated in Section Appendix A.2.4.

6.2.1. Coarse-to-fine input adaptation

In this case, the models has been trained on coarse resolution with N_S time steps, but at deployment, we are given fine-resolution target data with N_T time steps where $N_T = \bar{\rho} \times N_S$ with $\bar{\rho} \in \mathbb{Z}$. Hence we want to downsample the target dataset by a factor of $\bar{\rho}$ so that it matches the number of time steps in the source data and mimics the coarse resolution. Specifically, we consider the following input transformations:

Table 7
Performance of input adaptation for Course-to-Fine experiments on the SHD dataset.

Input Adapt.	$b_S = 2$ to $b_T = 1$ adLIF	$b_S = 2$ to $b_T = 1$ RadLIF
Binary-sum-bin	36.6 ± 2.1 %	40.6 ± 2.5 %
Max-pool	83.2 ± 1.0 %	85.2 ± 1.1 %

- Binary-sum-binning Input Adaptation Method: This method, similar to the sum-binning in Section 5.1.4, sums every $\bar{\rho}$ consecutive elements using a moving window (non-overlapping) and applies indicator function to the sum, hence for $\bar{\rho} = 2$, the sequence (x, y, z, w) becomes $(I(x + y), I(z + w))$ where I is the indicator function.
- Max-pooling Input Adaptation Method: This method compares every $\bar{\rho}$ consecutive elements using a non-overlapping moving window and replaces them with a single value representing the maximum of them. Hence, for $\bar{\rho} = 2$, the sequence (x, y, z, w) becomes $(\max(x, y), \max(z, w))$.

The results for these methods are presented in Table 7. Sum-binning (as used during training) can also be viewed as one of the input adaptation methods. Hence, input adaptation via sum-binning corresponds to the case already covered in Table 3, where the source and target data are identical. This provides a baseline performance when the method for obtaining coarse resolution data may be matched during training and test. Recall from Table 4(a) that for the case $b_S = 2$ to $b_T = 1$ using model adaptation yielded performance of 89.5 % for adLIF and 87.7 % for RadLIF. Comparing these results with the ones in Table 7, we observe that Integral and Expectation model adaptation methods outperform the Binary-sum-bin and Max-pool input adaptation methods.

6.2.2. Fine-to-coarse input adaptation

In this case, the models have been train on fine resolution with N_S time steps, but at deployment, we are given coarse-resolution target data with N_t time steps where $N_S = \rho \times N_T$ with $\rho \in \mathbb{Z}$. Hence we want to upsample the target dataset by a factor of ρ so that it matches the number of time steps in the source data and thus mimics the fine resolution. Specifically, we consider the following input transformations:

- Pad-zeros Input Adaptation Method: This method prefixes each element in the sequence with $\rho - 1$ zeros, hence for $\rho = 2$, the sequence (x, y, z, w) becomes $(0, x, 0, y, 0, z, 0, w)$.
- Repeat-elements Input Adaptation Method: This method repeats each element in the sequence $\rho - 1$ additional times, hence for $\rho = 2$, the sequence (x, y, z, w) becomes (x, x, y, y, z, z, w, w).

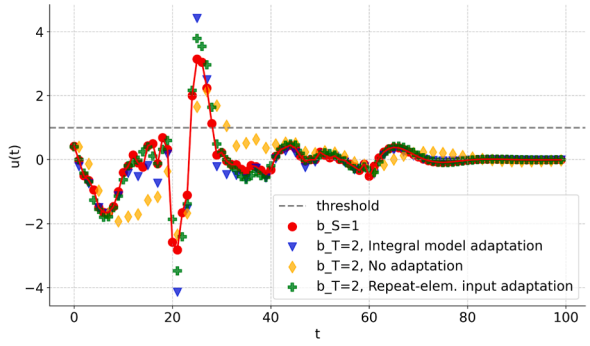
The results of input adaptation for the Fine-to-Coarse setting are presented in Table 8. Recall from Table 5(a) that for the case $b_S = 1$ to $b_T = 2$ using model adaptation yielded performance of 80.1 % for adLIF and 76.0 % for RadLIF. Comparing input adaptation with model adaptation, Repeat-elem method for adLIF; and Pad-zero and Repeat-elem methods for RadLIF surpass the performance achieved through model adaptation. However, it is important to note that although input adaptation shows better performance than model adaptation, it comes with the drawback of longer processing times since a longer time-sequence has to be processed in order to obtain the classification result. For instance, when going from $b_S = 1$ to $b_T = 2$, using the input-adaptation method will double the inference time since the length of the sequence in target domain after input adaptation is $2 \times N_T$ compared to using the proposed method where the length of the target sequence remains N_T.

6.2.3. Neuron dynamics within the network

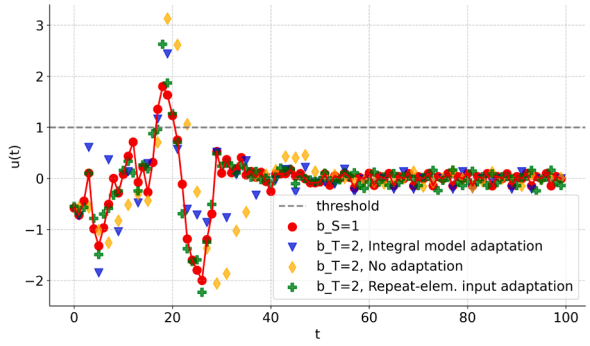
To gain insights on the dynamics of the neurons under model and input adaptation, we now consider a neuron inside one of the networks used in this paper's experiments. In particular, we consider a model

Table 8
Performance of input adaptation methods for Fine-to-Coarse experiments on the SHD dataset.

	$b_S = 1$ to $b_T = 2$	
Input Adapt.	adLIF	RadLIF
Pad-zeros	72.0 ± 4.1 %	88.9 ± 0.9 %
Repeat-elem	89.3 ± 0.7 %	89.9 ± 0.4 %



(a) Neuron dynamics for a single neuron in the first hidden layer for one input sample of the SHD data.



(b) Neuron dynamics for a single neuron in the second hidden layer for one input sample of the SHD data.

Fig. 5. Example of voltage dynamics over time in a pre-trained network on $b_S = 1$. Figure shows voltage dynamics of Fine-to-Coarse $b_S = 1$ to $b_T = 2$ experiment for different adaptation methods and time resolutions.

trained on the SHD dataset with $b_T = 1$, and examine how it behaves when operating under $b_T = 2$. Specifically, we take a sample from the SHD dataset and inspect the activity of one randomly selected neuron from the first hidden layer and one from the second hidden layer in Fig. 5.

Fig. 5 compares our Integral model adaptation, which processes coarse-resolution input, with the Repeat-elements input adaptation, which mimics fine-resolution data. As shown in Fig. 5(a), both the reference dynamics and the Repeat-elem method produce four output spikes, while the Integral method produces only two. Similarly, in Fig. 5(b), the reference and Repeat-elem generate three spikes, while the Integral method again outputs two. These observations indicate that while both methods capture the main peaks in the neuron variable $uNoArg[t]$, the Integral method is limited by the coarser temporal resolution, resulting in a lower number of spikes. In contrast, Repeat-elem operates on a longer sequence, providing more flexibility to match the finer details of the spiking pattern-reflected in the varying number of spikes across the two scenarios.

Table 9
Baseline performance $b_S = 2$ to $b_T = 2$ on the SHD dataset for various Coarse data transformation.

Coarse Transformation	adLIF	RadLIF
Sum-bin	91.0 ± 0.5 %	91.0 ± 0.7 %
Average-sum-bin	90.6 ± 0.5 %	90.8 ± 0.8 %
First-point-sample	87.3 ± 0.7 %	87.8 ± 0.6 %
Last-point-sample	88.3 ± 0.7 %	89.0 ± 0.9 %
Max-pool	90.7 ± 0.7 %	90.2 ± 0.5 %
Binary-sum-bin	87.9 ± 0.7 %	89.1 ± 0.3 %

Table 10
Performance on the SHD dataset under coarse data obtained with varying transformations.

(a) Fine-to-Coarse $b_S = 1$ to $b_T = 2$.

	Integral		No-adaptation	
Coarse Transformation	adLIF	RadLIF	adLIF	RadLIF
Sum-bin	75.4 ± 2.3 %	75.8 ± 2.8 %	57.1 ± 1.7 %	45.3 ± 2.5 %
Average-sum-bin	75.4 ± 2.3 %	75.8 ± 2.8 %	57.1 ± 1.7 %	45.3 ± 2.5 %
First-point-sample	69.6 ± 2.3 %	72.0 ± 2.9 %	55.3 ± 1.3 %	46.1 ± 2.1 %
Last-point-sample	68.7 ± 3.0 %	72.3 ± 2.7 %	55.5 ± 1.2 %	44.7 ± 2.3 %
Max-pool	54.5 ± 12.1 %	57.2 ± 9.5 %	41.1 ± 4.5 %	34.6 ± 4.0 %
Binary-sum-bin	28.0 ± 4.8 %	25.9 ± 2.3 %	19.7 ± 3.6 %	17.2 ± 2.4 %

(b) Coarse-to-Fine $b_S = 2$ to $b_T = 1$.

	Integral		No-adaptation	
Coarse Transformation	adLIF	RadLIF	adLIF	RadLIF
Sum-bin	89.6 ± 0.6 %	89.2 ± 0.7 %	51.6 ± 1.3 %	32.8 ± 2.7 %
Average-sum-bin	89.3 ± 0.6 %	89.5 ± 1.2 %	51.0 ± 1.4 %	33.8 ± 2.1 %
First-point-sample	86.3 ± 0.6 %	87.0 ± 1.2 %	47.0 ± 1.0 %	25.9 ± 6.3 %
Last-point-sample	87.9 ± 1.0 %	87.4 ± 1.1 %	45.3 ± 1.3 %	23.8 ± 4.1 %
Max-pool	73.7 ± 1.2 %	68.4 ± 6.4 %	38.9 ± 1.8 %	17.2 ± 4.5 %
Binary-sum-bin	60.0 ± 2.9 %	53.0 ± 2.2 %	33.5 ± 2.2 %	24.3 ± 2.4 %

6.3. Transformations for obtaining coarse data

In the previous experiments, the sequences with coarse temporal resolution are obtained using sum-binning, see Section 5.1.4. In this section, we explore five alternative transformations for obtaining coarse temporal resolution sequences: Average-sum-bin, First-point-sample, Last-point-sample, Max-pool and Binary-sum-bin. Average-sum-bin is similar to sum-binning with an additional scaling of the coarse-resolution sequence by ρ. Hence, Average-sum-bin is a scaled version of sum-bin. The First-point-sample and Last-point-sample perform simple index sampling of the fine-resolution sequence by using the first or last index in each bin, respectively. The Max-pool and Binary-sum-bin are defined in Section 6.2.1. The normalization layers adaptation for all these transformations is presented in Section Appendix A.2.4.

As a baseline, we report the performance of SNN models trained and tested on the same time resolution, i.e., $b_S = b_T = 2$, under different coarse resolution methods in Table 9. The table shows an average performance in the range of 87.3 % – 91.0 % under all transformations.

We present the performance under the various coarse transformations using Integral method and no adaptation for Fine-to-Coarse adaptation and Coarse-to-Fine adaptation in Table 10(a) and (b), respectively. We observe that performance of adaptation depends on the data representation in coarse resolution, where the adaptation performs weaker under Max-pool and Binary-sum-bin compared to other transformations. Nevertheless, our proposed adaptation method performs significantly better than no-adaptation for all coarse transformations and for both adLIF and RadLIF. These experiments illustrate that the proposed methods may achieve higher performance compared to no-adaptation across different coarse temporal resolution transformations.

6.4. Spike sparsity

In this section, we evaluate the performance of the proposed adaptation methods across different spiking rates. As discussed in Section 4.2, a neuron in a SNN reduces to a linear SSM when there is no nonlinearity in the neuron, where the spiking function of (2b) is the primary source of nonlinearity. Hence, as neurons spike more frequently, the linear approximation is expected to become less accurate, which is expected to degrade the performance of the proposed methods. We now explore this phenomena.

We consider a single adLIF neuron with sinusoidal inputs, as introduced in Section 5.2. Same as in Section 5.2, we initialize 100 distinct adLIF neurons and generate 100 different sine inputs for each neuron, averaging the results across all trials. Again, we evaluate the match in dynamics for both Fine-to-Coarse for $b_S = 1$ and $b_T = 2$ and Coarse-to-Fine for $b_S = 2$ and $b_T = 1$. To control the spiking activity, hence the level of nonlinearity, we vary the spiking threshold: higher thresholds reduce spiking activity, whereas lower thresholds increase it. This effect is illustrated in Figs. 6(a) and 7(a), where the spike rate denotes the mean number of output spikes per neuron over the input sequence length. In these figures, we report results for both randomly initialized neurons with b_S source temporal resolution input sequences, and for the adapted neurons with b_T target temporal resolution input sequences.

Based on Figs. 6(a) and 7(a), we now discuss how the spike rate changes under model adaption. We observe that for both the Coarse-to-Fine and Fine-to-Coarse cases, the no-adaptation baseline and all three proposed methods yield to spike rates similar to the spike rate obtained with inputs at the source resolution. In contrast, the Time-const. baseline produces relatively different spike-rate values. However, as shown earlier in Table 5(a), the Time-const. method can still outperform no-adaptation. This indicates that the spike-rate is not necessarily a good indicator of the performance of the adaptation methods.

Next, we discuss the match of neuron dynamics of the original and adapted neurons at different spiking thresholds; and hence different spiking levels. As in Section 5.2, we use the quality functions Q_1 and Q_2, where values close to 1 indicate better reconstruction. Results for the Coarse-to-Fine case are shown in Fig. 6(b) and (c), and for the Fine-to-Coarse case in Fig. 7(b) and (c). Across all methods, higher thresholds generally lead to improved matches, while the no-adaptation baseline shows either no significant change or no trend that is consistent in all plots. The higher quality values for the proposed methods under higher thresholds are consistent with the fact that under lower spike rates, the neuron dynamics is closer to linear, leading to a better performance of our proposed methods. This effect is most evident at the threshold of 10^2 where the spiking rate drops to zero, and both $Q_1 \approx 1$ and $Q_2 \approx 1$.

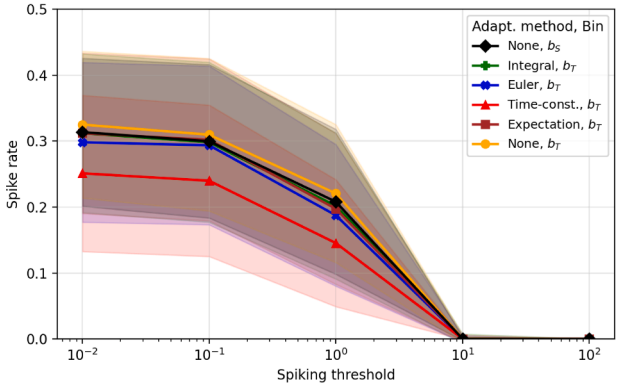
These experiments indicate that for the proposed adaptation methods, trained networks which have relatively lower spike rates may be more suitable. Hence, the performance of the proposed methods may be improved by preferring such networks while training on source data, e.g., by varying the spiking threshold or introducing regularization for spike rates.

6.5. Another linearization of neuron dynamics

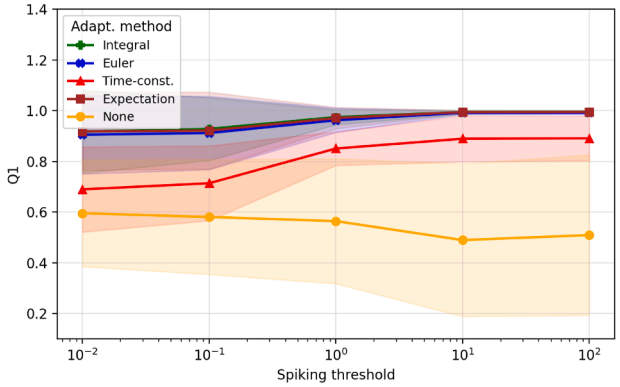
In the proposed adaptation methods of this article, we interpret the spike of the previous time step, i.e., $s_{out}[t]$, as an external input to the neuron state and use the correspondence $\boldsymbol{A} = [\boldsymbol{H}_v]$, and $\boldsymbol{B} = [\boldsymbol{H}_f, \boldsymbol{H}_i, \boldsymbol{H}_r]$, see Section 4.2. However, as pointed out in Section 4.2, an alternative approach is to linearize the spiking function $g_{\boldsymbol{\Theta}}(\cdot)$ and use $\boldsymbol{A} = \boldsymbol{H}_v + \boldsymbol{H}_f \boldsymbol{G}_{\boldsymbol{\Theta}}$, and $\boldsymbol{B} = [\boldsymbol{H}_i, \boldsymbol{H}_r]$. In this section, we explore this approach.

Linearization of $s_{out}[t] = g_{\boldsymbol{\Theta}}(\boldsymbol{V}[t])$ of (2b) depends on the particular form of the function. We now focus on the adLIF neuron with $s_{out}[t] = g_\theta(u[t])$ where $g_\theta(\cdot)$ is the Heaviside function with threshold θ, and $u[t]$ is the first component of the state vector $\boldsymbol{V}[t]$, see (1) and (4). To linearize $g_\theta(u[t])$, one may use different approaches. Here, we adopt

(a) Spike rate for both source and target resolution data



(b) Q_1 between source and target dynamics



(c) Q_2 between source and target dynamics

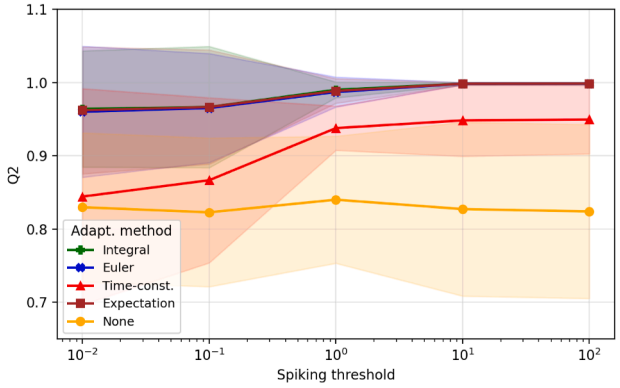


Fig. 6. Coarse-to-Fine scenario for $b_S = 2$ and $b_T = 1$. The solid line indicates the mean, and the shaded region represents one standard deviation around the mean.

$s_{out}[t] \approx 0.5u[t]$. Hence, we now have

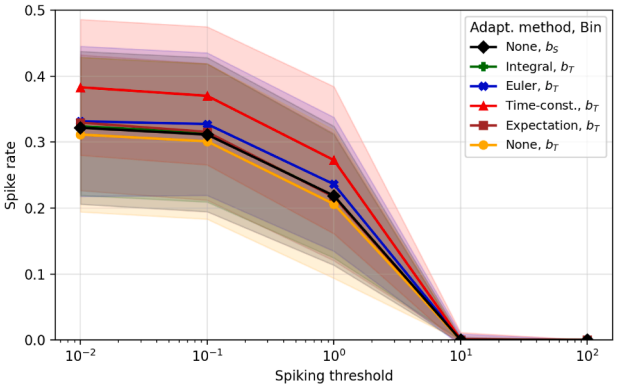
$$\boldsymbol{v}[t+1] \approx \boldsymbol{H}_v \boldsymbol{v}[t] + \boldsymbol{H}_f \begin{bmatrix} 0.5, & 0 \end{bmatrix} \boldsymbol{v}[t] + \begin{bmatrix} \boldsymbol{H}_i, \boldsymbol{H}_r \end{bmatrix} \begin{bmatrix} \mathbf{W} \boldsymbol{s}_{in}[t] \\ \boldsymbol{V} \boldsymbol{s}_{out}[t] \end{bmatrix}. \tag{21}$$

Using (4) and (21), we obtain the following mapping instead of the mapping in (15)

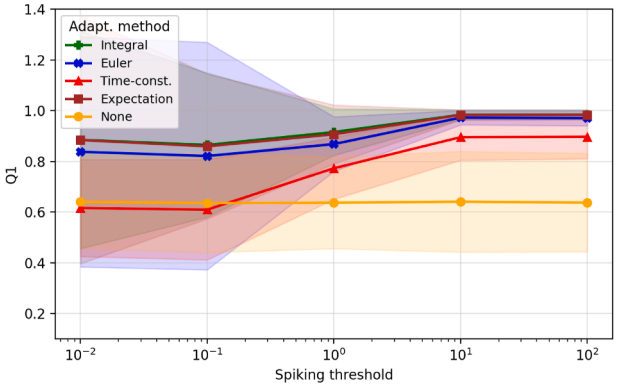
$$\boldsymbol{A} = \begin{bmatrix} \alpha - 0.5\alpha\theta & -(1-\alpha) \\ a + 0.5b & \beta \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1-\alpha & 1-\alpha \\ 0 & 0 \end{bmatrix}. \tag{22}$$

Table 11 presents the performance under (22) and application of the adaptation methods using Section Appendix A.1 for the SHD dataset.

(a) Spike rate for both source and target resolution data



(b) Q_1 between source and target dynamics



(c) Q_2 between source and target dynamics

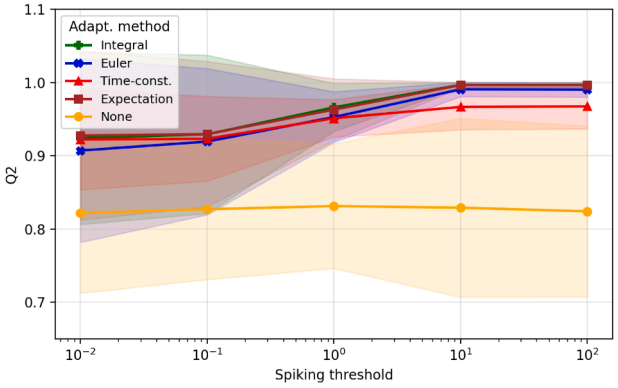


Fig. 7. Fine-to-Coarse scenario for $b_S = 1$ and $b_T = 2$. The solid line indicates the mean, and the shaded region represents one standard deviation around the mean.

Firstly, we discuss the Coarse-to-Fine scenario ($b_S = 2$ to $b_T = 1$), and compare with Table 4(a). We observe that there is a significant drop in performance of all three adaptation approaches under the alternative linearization in (22). For example, a drop from 89.5 % to 66.1 % is observed for the Integral method. However, even with this performance drop, the Exponential and Integral method still outperform both the no-adaptation scenario and Time-const method. Now we focus on the Fine-to-Coarse scenario ($b_S = 1$ to $b_T = 2$), and compare the results of Table 11 with the results of Table 5(a). We again observe a drop of performance under the alternative linearization of (22). Here, in some cases

Table 11
Performance of adaptation methods on the SHD dataset under alternative linearization of neuron dynamics.

	$b_S = 2$ to $b_T = 1$		$b_S = 1$ to $b_T = 2$	
Model Adapt.	adLIF	RadLIF	adLIF	RadLIF
Expectation	66.1 ± 2.6 %	41.4 ± 9.4 %	25.5 ± 3.9 %	21.7 ± 2.2 %
Integral	66.1 ± 2.6 %	41.4 ± 9.4 %	25.5 ± 3.9 %	21.7 ± 2.2 %
Euler	47.3 ± 3.2 %	17.1 ± 8.5 %	50.3 ± 4.5 %	44.1 ± 6.1 %

the performance of adaptation is even worse than that of no-adaptation and Time-const. method. We note that performance depends on the approach utilized for linearization of the non-linear spiking function; and other linearization approaches may improve the performance.

In conclusion, in the setting considered in this experiment, for both Coarse-to-Fine and Fine-to-Coarse scenarios, it is preferable to use the suggested correspondence in Section 4.2, i.e., treating the spike from the previous time step as external input to the neuron state as in (15), rather than the alternative linearization of (22).

7. Discussions

We now further discuss our results including the limitations of the proposed approaches.

As noted in Section 3.2, a variety of neuron models can be expressed within the generalized neuron model framework of Section (2), which allows application of our proposed methods. However, in this paper, our numerical results focus on the adLIF model, as it provides a good balance between the simplicity of the basic LIF model and the complexity of more detailed biophysical models, while also being widely used in the machine learning-oriented SNN literature, e.g. see Yik (2025). Nevertheless, investigating the performance of the proposed methods under other neuron models is a valuable direction for future research, as it will clarify the scope of their applicability in scenarios where models other than adLIF may be preferred.

As quantified in Sections 5.4 and 5.5, the greater the difference between the resolution of source and target, the less successful the proposed adaptation methods are. This is consistent with the fact that the mismatch between the characteristics of the source and target data increases with increasing ρ.

The proposed model adaptation methods consistently outperform the existing model parameter adaptation methods benchmarks in both Coarse-to-Fine and Fine-to-Coarse settings. In the case of Fine-to-Coarse settings, the input adaptation methods we propose can outperform the model adaptation methods. This comes at the cost of longer inference time that scales scales with the ratio of the resolutions and has to be performed for each sample seen during inference.

Our adaptation methods operate under the assumption that data is represented by events accumulated within fixed temporal windows, Δ_S for source data and Δ_T for target data. In such approaches, each time point in data sequence corresponds to a specific temporal interval used in accumulation of events. However, there are event representation formats where this relationship does not hold in a straightforward manner. For example, voxel-grid (Chen et al., 2023; Xu et al., 2025) may include weighted interpolation of a set of events in the time domain, and time-surface (Lagorce et al., 2016; Sironi et al., 2018) may encode recency via an exponential decay kernel at each pixel. In these cases, our methods are not directly applicable. An interesting avenue for future research is to investigate whether the use of intermediate fixed temporal-window based representations in the data processing chain, together with our adaptation methods, can yield satisfactory results for such cases.

An open question is the robustness of our adaptation methods to hardware non-idealities such as limited-precision representation of neuron parameters and synaptic weights, imperfect state updates, and general device noise. Since adaptation rules are derived under ideal device assumptions, their sensitivity to such effects remains to be studied.

Exploring these aspects and possible mitigation strategies constitute an important future research direction to enable practical applications of our framework.

8. Conclusions

We have established a bridge between SNNs and both linear and nonlinear SSMs. We have formulated the problem of temporal domain adaptation for SNNs under an SSM framework. We have proposed novel temporal domain adaptation methods using insights from the SSM framework. These methods do not require access to data with the target resolution. The proposed methods significantly outperformed the existing primary approach in the current literature for temporal domain model adaptation for SNNs for all scenarios where the model was trained on data with coarse temporal resolution and used on data with fine temporal resolution.

The proposed methods can be applied to any SNN neuron model. This addresses an important technical gap for time resolution adaptation in SNNs. In particular, the existing method in the literature is best suited to neuron models with explicit dependence on the temporal resolution, which is satisfied only for simpler neuron models.

Our results show that the proposed methods can be used to enable time-efficient training. In particular, high accuracy on data with fine temporal resolution can be obtained by first training on coarse temporal resolution data, hence with a lower training time, and then performing model adaptation using the proposed methods.

Our findings illustrate that working with datasets with static origins, e.g. NMNIST, provides limited insight into importance of time resolution, since such datasets do not have the temporal complexity inherent in real-world dynamic data. Hence, considering audio datasets where temporal dynamics play an important role, our work provides a significant contribution to the literature on temporal resolution adaptation in contrast to approaches that focus solely on data with static origin.

The starting point for our framework is to match the neuron voltage dynamics on a single neuron level across different temporal resolutions. Investigation of spike-oriented approaches that focus on matching the spiking behavior on a layer level is considered a promising direction for future work. Other interesting research directions include exploring the performance of our proposed methods with other popular neuron models, developing specialized temporal model adaptation techniques for other popular neuron model families, and evaluating the performance on non-ideal hardware.

CRediT authorship contribution statement

Sanja Karilanova: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization; **Maxime Fabre:** Writing – review & editing, Visualization, Investigation, Formal analysis, Conceptualization; **Emre Neftci:** Writing – review & editing, Visualization, Supervision, Investigation, Formal analysis, Conceptualization; **Ayça Özçelikkale:** Writing – review & editing, Visualization, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by Swedish Research Council through grant agreement no. 2024-05194. S. Karilanova and A. Özçelikkale acknowledge the support of Centre for Interdisciplinary Mathematics (CIM), and AI4Research, both at Uppsala University.

The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

We thank Prof. Mikael Sternad from Uppsala University for his invaluable feedback on the manuscript and insightful discussions which contributed to the improvement of this work. We thank Petra Leferink for her parallel master's thesis on fine-tuning SNNs for domain adaptation, and Prof. Elisabetta Chicca for her insightful supervision.

Appendix A

A.1. Propositions for proposed temporal resolution domain adaptation methods

We now present our temporal-resolution domain adaptation methods as propositions and their proofs. In this section, the methods are developed using the standard linear SSM notation for the sake of clarity. In Section 4.3, we present them using the notation of the generalized SNN neuron model in (2) based on the correspondence drawn in Section 4.2.

We now provide some preliminaries. A continuous-time linear SSM can be represented as (Gajic, 2003):

$$\frac{d}{dt}\boldsymbol{v}(t) = \boldsymbol{A}_c\boldsymbol{v}(t) + \boldsymbol{B}_c\boldsymbol{f}(t) \tag{A.1a}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}c\boldsymbol{v}(t) + \boldsymbol{D}c\boldsymbol{f}(t) \tag{A.1b}$$

where the vectors $\boldsymbol{v}(t), \boldsymbol{f}(t), \boldsymbol{y}(t)$ denote the state vector, input vector and output vector, respectively. Analogous to the discrete SSM defined in (11), the matrix $\boldsymbol{A}_c \in \mathbb{R}^{n\times n}$ describes the internal behaviour of the system, while matrices $\boldsymbol{B}_c \in \mathbb{R}^{n\times r}, \boldsymbol{C}c \in \mathbb{R}^{p\times n}, \boldsymbol{D}c \in \mathbb{R}^{p\times r}$ represent connections between the input and the system, and the output.

We now define two methods from the literature used to approximate the continuous SSM defined in (A.1) under a sampling period T_r with the discrete SSM defined in (11): the Integral Approximation method, and the Euler method. The aim of both discretization methods is to find $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ matrices in (11) such that the dynamics of the linear SSM and the continuous SSM are closely aligned. For clarity in this appendix, we denote these matrices with the subscript r, i.e. $\boldsymbol{A}_r, \boldsymbol{B}_r, \boldsymbol{C}_r, \boldsymbol{D}_r$ instead of $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$.

The Integral Approximation method for discretization is based on the assumption that the system's input is constant during the given sampling period. Hence, a discrete-time linear SSM is obtained such that (Gajic, 2003):

$$\boldsymbol{A}_r = e^{\boldsymbol{A}_c T_r} \tag{A.2a}$$

$$\boldsymbol{B}_r = (\boldsymbol{A}_r - \boldsymbol{I})\boldsymbol{A}_c^{-1}\boldsymbol{B}_c \tag{A.2b}$$

$$\boldsymbol{C}_r = \boldsymbol{C}c \tag{A.2c}$$

$$\boldsymbol{D}_r = \boldsymbol{D}c, \tag{A.2d}$$

which is valid under the assumption that $\boldsymbol{A}_c$ is invertible.

The Euler method for discretization is known as a less accurate but simpler method compared to the integral approximation method (Gajic, 2003). Euler method is based on the approximation of the first derivative at the time instant $t = kT_r$ (Gajic, 2003):

$$\frac{d}{dt}\boldsymbol{v}(t) \approx \frac{1}{T_r}\big(\boldsymbol{v}((k+1)T_r) - \boldsymbol{v}(kT_r)\big). \tag{A.3}$$

Applying this approximation of the derivative, a discrete-time linear SSM is obtained such that (Gajic, 2003):

$$\boldsymbol{A}_r = \boldsymbol{I} + T_r \cdot \boldsymbol{A}_c \tag{A.4a}$$

$$\boldsymbol{B}_r = T \cdot \boldsymbol{B}_c \tag{A.4b}$$

$$\boldsymbol{C}_r = \boldsymbol{C}_c \tag{A.4c}$$

$$\boldsymbol{D}_r = \boldsymbol{D}_c. \tag{A.4d}$$

Assumption 1. Consider two discrete-time linear SSM (11), with matrix parameters $\boldsymbol{A}_{r1}, \boldsymbol{B}_{r1}, \boldsymbol{C}_{r1}, \boldsymbol{D}_{r1}$, and $\boldsymbol{A}_{r2}, \boldsymbol{B}_{r2}, \boldsymbol{C}_{r2}, \boldsymbol{D}_{r2}$, with associated

time resolution T_{r1} and T_{r2}, respectively. Assume $T_{r2} = \rho T_{r1}$ for $\rho \in \mathbb{R}_+$. Additionally we assume $\boldsymbol{C}_{r1} = \boldsymbol{C}_{r2}$, and $\boldsymbol{D}_{r1} = \boldsymbol{D}_{r2}$.

Proposition 1. ***Integral temporal adaptation method:*** *Consider two discrete-time linear SSM denoted with subscript r_1 and r_2 as defined in Assumption 1 obtained from the continuous-time linear SSM (A.1) using the Integral Approximation method with sampling periods T_{r2} and T_{r1}, respectively. Assume that $(\boldsymbol{A}_{r1} - \boldsymbol{I})$ is invertible, then:*

$$\boldsymbol{A}_{r2} = \boldsymbol{A}_{r1}^{\rho} \tag{A.5a}$$

$$\boldsymbol{B}_{r2} = (\boldsymbol{A}_{r2} - \boldsymbol{I})(\boldsymbol{A}_{r1} - \boldsymbol{I})^{-1}\boldsymbol{B}_{r1} \tag{A.5b}$$

Proof. of Proposition 1: By (A.2a) and (A.2b) that

$$\boldsymbol{A}_{r2} = e^{\boldsymbol{A}_c T_{r2}} = e^{\boldsymbol{A}_c \rho T_{r1}} = (e^{\boldsymbol{A}_c T_{r1}})^{\rho} = \boldsymbol{A}_{r1}^{\rho} \tag{A.6a}$$

$$\boldsymbol{B}_{r2} = (\boldsymbol{A}_{r2} - \boldsymbol{I})\boldsymbol{A}_c^{-1}\boldsymbol{B} = (\boldsymbol{A}_{r2} - \boldsymbol{I})(\boldsymbol{A}_{r1} - \boldsymbol{I})^{-1}\boldsymbol{B}_{r1}. \tag{A.6b}$$

□

Proposition 2. ***Euler temporal adaptation method:*** *Consider two discrete-time linear SSM denoted with subscript r_1 and r_2 as defined inAssumption 1 obtained from the continuous-time linear SSM (A.1) using the Euler method with sampling periods T_{r2} and T_{r1}, respectively. Then:*

$$\boldsymbol{A}_{r2} = \boldsymbol{I} + \rho(\boldsymbol{A}_{r1} - \boldsymbol{I}) \tag{A.7a}$$

$$\boldsymbol{B}_{r2} = \rho \boldsymbol{B}_{r1} \tag{A.7b}$$

Proof. of Proposition 2: It follows by (A.4a) and (A.4b) that

$$\boldsymbol{A}_{r2} = \boldsymbol{I} + T_{r2}\boldsymbol{A}_c = \boldsymbol{I} + \rho T_{r1}\boldsymbol{A}_c \tag{A.8}$$

$$= \boldsymbol{I} + \rho((T_{r1}\boldsymbol{A}_c + \boldsymbol{I}) - \boldsymbol{I}) = \boldsymbol{I} + \rho(\boldsymbol{A}_{r1} - \boldsymbol{I}) \tag{A.9}$$

$$\boldsymbol{B}_{r2} = T_{r2}\boldsymbol{B}_c = \rho T_{r1}\boldsymbol{B}_c = \rho(T_{r1}\boldsymbol{B}_c) = \rho \boldsymbol{B}_{r1}. \tag{A.10}$$

□

Proposition 3. ***Expectation temporal adaptation method:*** *Consider two discrete-time linear SSM denoted with the subscripts r_1 and r_2 as defined in Assumption 1 with the input sequences $\boldsymbol{f}[n]$ for $n \in 1, 2, \dots, N_{r1}$, and $\bar{\boldsymbol{f}}[m]$ for $m \in 1, \dots, N_{r2}$, respectively. Here, $\bar{\rho} N_{r_2} = N_{r_1}, \bar{\rho} \in \mathbb{Z}_+$. Assume that $\mathbb{E}[\bar{\boldsymbol{f}}[m]] = \mathbb{E}[\boldsymbol{f}[(m+1)\bar{\rho} - k]] = \mu_m$ for $k = 0, \dots, \bar{\rho} - 1$ for all m. Then, the following holds:*

1. *If*

$$\boldsymbol{A}_{r2} = \boldsymbol{A}_{r1}^{\bar{\rho}} \tag{A.11a}$$

$$\boldsymbol{B}_{r2} = \sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r1}^{\bar{\rho}-j}\boldsymbol{B}_{r1}, \tag{A.11b}$$

   then $\mathbb{E}[\boldsymbol{v}_{r2}[m]] = \mathbb{E}[\boldsymbol{v}_{r1}[m\bar{\rho}]]$ for all m.
2. *Assume that $(\sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r2}^{\bar{\rho}-j})$ is invertible. If*

$$\boldsymbol{A}_{r1} = \boldsymbol{A}_{r2}^{\frac{1}{\bar{\rho}}} \tag{A.12a}$$

$$\boldsymbol{B}_{r1} = (\sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r1}^{\bar{\rho}-j})^{-1}\boldsymbol{B}_{r2} \tag{A.12b}$$

   then $\mathbb{E}[\boldsymbol{v}_{r2}[m]] = \mathbb{E}[\boldsymbol{V}_{r1}[m\bar{\rho}]]$ for all m.

Proof. of Proposition 3 We first express $\mathbf{v}_{r1}[n + \bar{\rho}]$ in terms of $\mathbf{v}_{r1}[n]$ by performing $\bar{\rho}$ recursive steps:

$$\mathbf{v}_{r1}[n + \bar{\rho}] = \boldsymbol{A}_{r1}\mathbf{v}_{r1}[n + \bar{\rho} - 1] + \boldsymbol{B}_{r1}\boldsymbol{f}[n + \bar{\rho} - 1] \tag{A.13}$$

$$\mathbf{v}_{r1}[n + \bar{\rho}] = \boldsymbol{A}_{r1}(\boldsymbol{A}_{r1}\mathbf{v}_{r1}[n + \bar{\rho} - 2] + \boldsymbol{B}_{r1}\boldsymbol{f}[n + \bar{\rho} - 2]) + \boldsymbol{B}_{r1}\boldsymbol{f}[n + \bar{\rho} - 1]$$

$$\vdots \tag{A.14}$$

$$\mathbf{v}_{r1}[n + \bar{\rho}] = \boldsymbol{A}_{r1}^{\bar{\rho}}\mathbf{v}_{r1}[n] + \sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r1}^{j-1}\boldsymbol{B}_{r1}\boldsymbol{f}[n + \bar{\rho} - j] \tag{A.15}$$

We now consider the constraint on the expected dynamics of the state variable i.e., $\mathbb{E}[\boldsymbol{V}_{r2}[m+1]] = \mathbb{E}[\boldsymbol{V}_{r1}[(m+1)\bar{\rho}]]$. Rewriting both sides of the equation, we obtain

$$\mathbb{E}\left[\mathbf{v}_{r2}[m+1]\right] = \mathbb{E}[\mathbf{v}_{r1}[(m+1)\bar{\rho}]] \tag{A.16}$$

$$\mathbb{E}\left[\boldsymbol{A}_{r2}\mathbf{v}_{r2}[m] + \boldsymbol{B}_{r2}\boldsymbol{f}\,bar[m]\right] =$$

$$= \mathbb{E}\left[\boldsymbol{A}_{r1}^{\bar{\rho}}\mathbf{v}_{r1}[m\bar{\rho}] + \sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r1}^{j-1}\boldsymbol{B}_{r1}\boldsymbol{f}[(m+1)\bar{\rho} - j]\right] \tag{A.17}$$

$$\boldsymbol{A}_{r2}\mathbb{E}\left[\mathbf{v}_{r2}[m]\right] + \boldsymbol{B}_{r2}\boldsymbol{\mu}_{\boldsymbol{m}} =$$

$$= \boldsymbol{A}_{r1}^{\bar{\rho}}\mathbb{E}\left[\mathbf{v}_{r1}[m\bar{\rho}]\right] + \sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r1}^{\bar{\rho}-j}\boldsymbol{B}_{r1}\boldsymbol{\mu}_{\boldsymbol{m}} \tag{A.18}$$

Hence, if (A.11) holds, then $\mathbb{E}[\boldsymbol{V}_{r2}[m]] = \mathbb{E}[\boldsymbol{V}_{r1}[m\bar{\rho}]]$ and $\mathbb{E}[\boldsymbol{V}_{r2}[m+1]] = \mathbb{E}[\boldsymbol{V}_{r1}[(m+1)\bar{\rho}]]$. Due to the generality of the previous equations, this holds for all $m \in 1, \dots, N_{r2}$. Similiarly, if $(\sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r2}^{\bar{\rho}-j})$ is invertible and (A.12) holds, we have $\mathbb{E}[\boldsymbol{V}_{r2}[m]] = \mathbb{E}[\boldsymbol{V}_{r1}[m\bar{\rho}]]$ for all m. This concludes the proof of Proposition 3. □

Remark 4. In Section 4.3, we use (A.11) and (A.12), for adaptation in Fine-to-Coarse and Coarse-to-Fine scenarios, respectively. These equations are presented separately in Proposition 3 for clarity.

Remark 5. We now compare Proposition 3, i.e. Expectation temporal adaptation method, and Proposition 1, i.e. Integral temporal adaptation method, in terms of their setting. Proposition 3 assumes constant expected value of the input over an interval i.e. $E[\boldsymbol{f}[(m+1)\bar{\rho} - k]] = \boldsymbol{\mu}_{\boldsymbol{m}}$ for $k = 0, 1, ., \bar{\rho}$ for some arbitrary $\boldsymbol{\mu}_{\boldsymbol{m}} \in \mathbb{R}^{1\times 1}$. Given this assumption, its goal is to match the expected values of the state variable at the time points of the coarse resolution, i.e. $\mathbb{E}[\boldsymbol{V}_{r2}[m]] = \mathbb{E}[\boldsymbol{V}_{r1}[m\bar{\rho}]]$. On the other hand, the integral approximation method which lies under Proposition 1, assumes that the input is constant over the interval i.e. $\boldsymbol{f}[(m+1)\bar{\rho} - k] = c_b$ for $k = 0, 1, ., \bar{\rho}$ for some arbitrary $c_b \in \mathbb{R}^{1\times 1}$ (Gajic, 2003). Given this assumption, Proposition 1 matches the values of the state variable at the time points of the coarse resolution, i.e. $\boldsymbol{V}_{r2}[m] = \boldsymbol{V}_{r1}[m\bar{\rho}]$. Note that matching the input and state variable implies matching the average behaviour of the input and the state variable. Hence, the assumptions and the goals in the Integral temporal adaptation method imply the assumptions and the goals in the Expectation temporal adaptation method.

Remark 6. Despite the different starting points for Expectation method, i.e., Proposition 3, and Integral method, i.e., Proposition 2, the two methods are equivalent under the below assumptions:

Fine-to-Coarse deployment: Integral method makes the assumption that $\boldsymbol{A}_{r1} - \boldsymbol{I}$ is invertible, while Expectation method does not make such an assumption. However, if $\boldsymbol{A}_{r1} - \boldsymbol{I}$ is also assumed to be invertible under Expectation method, then the two methods are equivalent.

Coarse-to-Fine deployment: Integral method makes the assumption that $\boldsymbol{A}_{r1} - \boldsymbol{I}$ is invertible, while Expectation method assumes $(\sum_{j=1}^{\bar{\rho}} \boldsymbol{A}_{r2}^{\bar{\rho}-j})$ is invertible. The two methods become equivalent if in the Expectation method, one also makes assumption that $\boldsymbol{A}_{r1} - \boldsymbol{I}$ and $\boldsymbol{A}_{r2} - \boldsymbol{I}$ is invertible.

Note that these equivalences are due to the fact that $\sum_{j=1}^{\bar{\rho}} \boldsymbol{A}^{\bar{\rho}-j} = (\boldsymbol{A}^{\bar{\rho}} - \boldsymbol{I})(\boldsymbol{A} - \boldsymbol{I})^{-1}$ under invertible $(\boldsymbol{A} - \boldsymbol{I})$, $\bar{\rho} \in \mathbb{Z}_+$.

A.2. Temporal domain adaptation for the normalization layers

During training of the neural networks, normalization layers learn the statistics, i.e. the mean and variance, of source data, which are then typically used in inference without any change. On the other hand, in our setup, the model is used on target data, which has a different time resolution. Hence, the statistics learned during training should be adjusted. In this section, we investigate how this adjustment should be made.

Denote a source sequence with $X_S = (x_1, x_2, \dots, x_{N_S})$ and the associated mean with $\mu_S = \frac{1}{N_S}\sum_{i=1}^{N_S} \mathbb{E}[x_i]$ and the variance with $\sigma_S^2 =$

$\frac{1}{N_S}\sum_{i=1}^{N_S}\mathbb{E}[(x_i-\mathbb{E}[x_i])^2]$. Similarly, denote a target sequence with $X_T = (x'_1, x'_2, \dots, x'_{N_T})$ and the associated mean and variance with μ_T and σ_T^2, respectively. Denote the ratio of the temporal resolutions with $\bar{\rho} = N_T/N_S$. Assume that the elements of source sequence, i.e., x_i are i.i.d.

A.2.1. Adaptation for sum binning transformation

Let the target sequence be the sum-binned version of the source sequence i.e. $N_S = \bar{\rho}N_T$ with $\bar{\rho} \in \mathbb{Z}_+$. Then, under i.i.d. x_i, we have $\mu_T = \mathbb{E}[x'_i] = \mathbb{E}[\sum_{j=0}^{\bar{\rho}-1} x_{i+j}] = \bar{\rho}\mathbb{E}[x_i] = \bar{\rho}\mu_S$, and $\sigma_T^2 = var(x'_i) = var(\sum_{j=0}^{\bar{\rho}-1} x_{i+j}) = \bar{\rho}^2 var(x_i) = \bar{\rho}^2\sigma_S^2$.

A.2.2. Adaptation for repeat elements transformation

Let the target sequence be obtained by repeating each element of the source sequence $\bar{\rho}-1$ times. Then, $\mu_T = \mathbb{E}[x'_i] = \mathbb{E}[x_i] = \mu_S$, and $\sigma_T^2 = var(x'_i) = var(x_i) = \sigma_S^2$.

A.2.3. Adaptation for zero padding transformation

Let the target sequence be obtained by adding $\bar{\rho}-1$ zeros before each element of the source sequence. Then, $\mu_T = E[x'_i] = \frac{x_1+\dots+x_N}{\bar{\rho}N} = \frac{\mu_S}{\bar{\rho}}$, and using the identity $var(x'_i) = E[x'^2_i] - (E[x'_i])^2$

$$\sigma_T^2 = \frac{1}{\bar{\rho}N_S}\sum_{i=1}^{\bar{\rho}N_S}\mathbb{E}[x'^2_i] - \frac{1}{\bar{\rho}N_S}\sum_{i=1}^{\bar{\rho}N_S}(\mathbb{E}[x'_i])^2 \tag{A.19}$$

$$= \frac{1}{\bar{\rho}N_S}\sum_{i=1}^{\bar{\rho}N_S}\mathbb{E}[x'^2_i] - (\frac{\mu_S}{\bar{\rho}})^2 \tag{A.20}$$

$$= \frac{1}{\bar{\rho}}\frac{1}{N_S}\sum_{i=1}^{N_S}\mathbb{E}[(x_i)^2] - (\frac{\mu_S}{\bar{\rho}})^2 \tag{A.21}$$

$$= \frac{1}{\bar{\rho}}\left[\sigma_S^2+\mu_S^2\right] - (\frac{\mu_S}{\bar{\rho}})^2 \tag{A.22}$$

$$= \frac{1}{\bar{\rho}}\sigma_S^2 + \frac{\mu_S^2(\bar{\rho}-1)}{\bar{\rho}^2} \tag{A.23}$$

A.2.4. Temporal domain adaptation in experiments

We now present how adaptation of batch normalization in performed in the numerical experiments.

In the Coarse-to-Fine scenario, where the source data is the sum-binned data and the target data is the fine resolution data, we have $N_S = \bar{\rho}N_T$ with $\bar{\rho} < 1$. The adaptations are approximated using the transformations in Section A.2.1. Hence, we use $\mu_T = \bar{\rho}\mu_S$ and $\sigma_T^2 = \bar{\rho}^2\sigma_S^2$. In the input adaptation of Section 6.2.1, based on numerical experiments, we use $\mu_T = \bar{\rho}\mu_S, \sigma_T^2 = \bar{\rho}\sigma_S^2$ for the max-pooling input adaptation method and $\mu_T = \bar{\rho}\mu_S, \sigma_T^2 = \bar{\rho}^2\sigma_S^2$ for the binary sum-binning input adaptation method.

In the Fine-to-Coarse scenario, where the source data is the fine data and the target data is the sum-binned resolution data, we have exactly the scenario of Section A.2.1, and hence use $\mu_T = \bar{\rho}\mu_S$ and $\sigma_T^2 = \bar{\rho}^2\sigma_S^2$. In the Fine-to-Coarse experiments, for the input adaptation methods of Section 6.2.2, additional transformations are applied to the coarse target data such as repeat elements or zero padding, with the aim of mimicking fine resolution during inference. When the transformation for mimicking fine resolution is repeated elements, we use the scaling from the repeated elements (Section A.2.2), in addition to the sum binning (Section A.2.1). Hence, we obtain $\mu_T = \bar{\rho}\mu_S$ and $\sigma_T^2 = \bar{\rho}^2\sigma_S^2$. When the transformation for mimicking fine resolution is zero-padding, we have repeated elements (Section A.2.3), in addition to the sum binning (Section A.2.1). Hence, we obtain the theoretical scaling as $\mu_T = \bar{\rho}\mu_S$ and $\sigma_T^2 = \bar{\rho}\sigma_S^2 + \mu_S^2(\bar{\rho}-1)$. However, experiments showed poor performance with this theoretical scaling of the variance. Hence, we use $\sigma_T^2 = \sigma_S^2$ in our results in Section 5.

For the transformations introduced in Section 6.3, based on our numerical experiments, we use the following parameter correspondences: For binary-sum-bin and max-pool, we use $\mu_T = \bar{\rho}\mu_S$ and $\sigma_T^2 = \sigma_S^2$. For first-point-sample, last-point-sample, and average-sum-bin, we use $\mu_T = \mu_S$ and $\sigma_T^2 = \sigma_S^2$. These relations are employed in both Coarse-to-Fine and Fine-to-Coarse scenarios.

Note that, in both Coarse-to-Fine and Fine-to-Coarse scenarios, we only adapt the first normalization layer in the network. This initial adjustment is considered to correct the shift introduced by the data transformation, reducing the need for additional scaling in subsequent layers.

A.3. Performance criteria for assessing matching neuron dynamics under varying time resolution

Given time sequence $v_1 \in \mathbb{R}^{N_1}$ with N_1 number of time steps, and signal $v_2 \in \mathbb{R}^{N_2}$ with N_2 number of time steps where we have either $N_1 = \bar{\rho}N_2$ or $N_2 = \bar{\rho}N_1$ where $\bar{\rho} \in \mathbb{Z}_+^{1\times 1}+$. Our objective is to estimate the goodness of match between these two sequences of different temporal resolutions are describing assuming the v_1 is the true sequence. Since the performance criteria functions that we will use require the two signals we compare to be of same time length, we uniformly sample the higher time resolution signal by taking every $\bar{\rho}$-th sample. For demonstration we focus on the case where v_1 is of higher time resolution i.e. $N_1 = \bar{\rho}N_2$, hence we create the sampled signal $\hat{v}_1$ such that $\hat{v}_1[n] = v_1[\bar{\rho}n]$ for $n = 1, \dots, N_2$. Then the performance criteria functions are defined as

$$Q_1(\hat{v}_1, v_2) = 1 - \frac{\mathrm{MSE}(\hat{v}_1, v_2)}{\mathrm{Var}(\hat{v}_1)} \tag{A.24}$$

$$Q_2(\hat{v}_1, v_2) = \frac{\mathrm{Cov}(\hat{v}_1, v_2)}{\sqrt{\mathrm{Var}(\hat{v}_1)\mathrm{Var}(v_2)}} \tag{A.25}$$

where

$$\mathrm{MSE}(\hat{v}_1, v_2) = \frac{1}{N_2}\sum_{n=1}^{N_2}(\hat{v}_1[n] - v_2[n])^2 \tag{A.26}$$

$$\mathrm{Cov}(\hat{v}_1, v_2) = \frac{1}{N_2-1}\sum_{n=1}^{N_2}(\hat{v}_1[n] - \overline{\hat{v}_1})(v_2[n] - \overline{v_2}). \tag{A.27}$$

Here $\overline{\hat{v}_1}$ and $\overline{v_2}$ are the mean of $\hat{v}_1$ and v_2, respectively, and Var is the variance. Here, Q_1 gives the relative square error and Q_2 gives the correlation coefficient, hence both provide different measures of similarity between $\hat{v}_1$ and v_2. When there is significant matching between $\hat{v}_1$ and v_2, MSE is expected to be small compared to variance, and both Q_1 and Q_2 are expected to take values $|Q_1|, |Q_2| \leq 1$ where $Q_1 \approx 1$ and $Q_2 \approx 1$ indicates better reconstruction compared to $Q_1 \approx 0$ and $Q_2 \approx 0$, respectively. Specifically $Q_1 = 1$ corresponds to perfect match, while $Q_2 = 1$ corresponds to perfect linear relationship between the two sequences.

A.4. Hyperparameters

For the adLIF parameters, α, β, a, and b are uniformly initialized in the regions $[\exp(-\frac{1}{5}), -\frac{1}{25})]$, $[\exp(-\frac{1}{30}), -\frac{1}{120})]$, $[0, 1)$, and $[0, 2]$, respectively. The clipping ranges while training for a and b are the same as their initialization ranges, while for both α and β the clipping range used is $[0, 1)$. In the training process, we use Adam optimizer with weight-decay= 0.0001 and Step Learning Rate scheduler of base learning rate 0.01, step-size 10 and $\gamma = 0.1$.

References

Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., & Hafid, A. S. (2023). A comprehensive survey on tinyML. *IEEE Access*, *11*, 96892–96922. https://doi.org/10.1109/ACCESS.2023.3294111

Bae, M., Shin, Y., Nam, Y., Lee, Y. S., & Lee, J.-G. (2024). Semi-supervised learning for time series collected at a low sampling rate. In *Proceeding of the 30th ACM SIGKDD conference on knowledge discovery and data mining* (p. 59–70).

Bahi, A., Gasmi, I., Bentrad, S., & Khantouchi, R. (2025). Enhancing recommendation diversity in e-commerce using siamese network and cluster-based technique. *Bulletin of Electrical Engineering and Informatics*, *14*(2), 1223–1230.

Bal, M., & Sengupta, A. (2025). P-SpikeSSM: Harnessing Probabilistic Spiking State Space Models for Long-Range Dependency Tasks. In *The Thirteenth International Conference on Learning Representations (ICLR)* 1–16.

Baronig, M., Ferrand, R., Sabathiel, S., & Legenstein, R. (2025). Advancing spatio-temporal processing in spiking neural networks through adaptation. *Nature Communications 16*. https://doi.org/10.1038/s41467-025-60878-z

Bittar, A., & Garner, P. N. (2022). A surrogate gradient spiking baseline for speech command recognition. *Frontiers in Neuroscience*, *16*. https://doi.org/10.3389/fnins.2022.865897

Caccavella, C., Paredes-Vall'es, F., Cannici, M., & Khacef, L. (2024). Low-power event-based face detection with asynchronous neuromorphic hardware. *International joint conference on neural networks (IJCNN)*, (pp. 1–10).

Chakravarthi, B., Verma, A. A., Daniilidis, K., Fermuller, C., & Yang, Y. (2024). Recent event camera innovations: A survey. In *European conference on computer vision* (pp. 342–376). Springer.

Chen, H., Chung, V., Tan, L., & Chen, X. (2023). Dense voxel 3d reconstruction using a monocular event camera. arXiv preprint arXiv:2309.00385.

Chen, M., Gündüz, D., Huang, K., Saad, W., Bennis, M., Feljan, A. V., & Poor, H. V. (2021). Distributed learning in wireless networks: Recent progress and future challenges. *IEEE Journal on Selected Areas in Communications*, *39*(12), 3579–3605.

Chen, Y., Mai, Y., Feng, R., & Xiao, J. (2022). An adaptive threshold mechanism for accurate and efficient deep spiking convolutional neural networks. *Neurocomputing*, *469*, 189–197. https://doi.org/10.1016/j.neucom.2021.10.080

Cramer, B., Stradmann, Y., Schemmel, J., & Zenke, F. (2022). The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(7), 2744–2757. https://doi.org/10.1109/TNNLS.2020.3044364

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.-H., Wild, A., Yang, Y., & Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99.

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., Plank, P., & Risbud, S. R. (2021). Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceeding of the IEEE*, *109*(5), 911–934. https://doi.org/10.1109/JPROC.2021.3067593

Dieter, W. R., Datta, S., & Kai, W. K. (2005). Power reduction by varying sampling rate. In *Proceedings of the international symposium on low power electronics and design* (pp. 227–232).

Du, Y., Liu, X., & Chua, Y. (2024). Spiking structured state space model for monaural speech enhancement. In *proceedings under international conference on acoustics, speech, and signal processing (icassp)* (pp. 766–770). https://doi.org/10.1109/ICASSP48485.2024.10448152

Eshraghian, J. K., Ward, M., Neftci, E. O., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., & Lu, W. D. (2023). Training spiking neural networks using lessons from deep learning. *Procedding of the IEEE*, *111*(9), 1016–1054. https://doi.org/10.1109/JPROC.2023.3308088

Farahani, A., Voghoei, S., Rasheed, K., & Arabnia, H. R. (2021). A brief review of domain adaptation. In *Advances in data science and information of engineering* (pp. 877–894). Springer Inter. Publishing.

Gajic, Z. (2003). Linear dynamic systems and signals. Prentice Hall - Pearson Education Inc.

Gallego, G., Delbrück, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A. J., Conradt, J., Daniilidis, K., & Scaramuzza, D. (2022). Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*, 154–180.

Gehrig, D., & Scaramuzza, D. (2022). Are high-resolution event cameras really needed? arXiv preprint arXiv:2203.14672.

Gehrig, D., & Scaramuzza, D. (2024). Low-latency automotive vision with event cameras. *Nature*, *629*(8014), 1034–1040.

Gerstner, W., Werner, M. K., Naud, R., & Paninski, L. (2014). Neuronal dynamics: From single neurons to networks and models of cognition. Cambridge University Press.

Hammouamri, I., Khalfaoui-Hassani, I., & Masquelier, T. (2024). Learning delays in spiking neural networks using dilated convolutions with learnable spacings. In *International Conference on Learning Representations (ICLR)*.

He, W., Wu, Y., Deng, L., Li, G., Wang, H., Tian, Y., Ding, W., Wang, W., & Xie, Y. (2020). Comparing SNNs and RNNs on neuromorphic vision datasets: Similarities and differences. *Neural Networks*, *132*, 108–120. https://doi.org/10.1016/j.neunet.2020.08.001

Higham, N. J., & Al-Mohy, A. H. (2010). Computing matrix functions. *Acta Numerica*, *19*, 159–208.

Higham, N. J., & Lin, L. (2011). A schur-padÃ© algorithm for fractional powers of a matrix. *SIAM Journal on Matrix Analysis and Applications*, *32*(3), 1056–1078. https://doi.org/10.1137/10081232X

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, *117*, 500–544.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the international conference on machine learning* (p. 448–456). (*vol. 37*).

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*(6), 1569–1572.

Jain, A., & Chang, E. Y. (2004). Adaptive sampling for sensor networks. In *Proceeding of the 1st international workshop on data management for sensor networks, VLDB*.

Karilanova, S., Dey, S., & Özçelikkale, A. (2025a). Low-bit data processing using multiple-output spiking neurons with non-linear reset feedback. *IEEE Journal of Selected Topics in Signal Processing*, (pp. 1–15). https://doi.org/10.1109/JSTSP.2025.3595030

Karilanova, S., Dey, S., & Özçelikkale, A. (2025b). State-space model inspired multiple-input multiple-output spiking neurons. In *2025 neuro inspired computational elements (NICE)* (pp. 1–9). https://doi.org/10.1109/NICE65350.2025.11065909

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., & Benosman, R. B. (2016). Hots: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligencee*, *39*(7), 1346–1359.

Lenz, G., Chaney, K., ShresthaS. B., Oubari, O., Picaud, S., & Zarrella, G.(2021). Tonic: Event-based datasets and transformations. https://tonic.readthedocs.io, https://doi.org/10.5281/zenodo.5079802

Liang, J., He, R., & Tan, T. (2024). A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*. https://doi.org/10.1007/s11263-024-02181-w

Liu, M., & Delbruck, T. (2018). Adaptive time-slice block-matching optical flow algorithm for dynamic vision sensors. The British Machine Vision Conference (BMVC).

Ma, C., Chen, X., Li, Y., Yang, Q., Wu, Y., Li, G., Pan, G., Tang, H., Tan, K. C., & Wu, J. (2025). Spiking neural networks for temporal processing: Status quo and future prospects. arXiv preprint arXiv:2502.09449.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7

Mazumder, M., Chitlangia, S., Banbury, C., Kang, Y., Ciro, J. M., Achorn, K., Galvez, D., Sabini, M., Mattson, P., & Kanter, D. et al. (2024). Multilingual spoken words corpus. In *Conference on neural information processing systems (neurIPS)*.

Müller-Cleve, S. F., Fra, V., Khacef, L., PequeÃ±o-Zurro, A., Klepatsch, D., Forno, E., Ivanovich, D. G., Rastogi, S., Urgese, G., Zenke, F., & Bartolozzi, C. (2022). Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware. *Frontiers in Neuroscience*, *16*. https://doi.org/10.3389/fnins.2022.951164

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., & Catanzaro, B. et al. (2021). Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1–15).

Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, *36*, 51–63.

Okuyama, Y. (2014). Discrete control systems. Springer London, Limited.

Orchard, G., Jayawant, A., Cohen, G. K., & Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, *9*, 437.

Park, J., Samarakoon, S., Elgabli, A., Kim, J., Bennis, M., Kim, S.-L., & Debbah, M. (2021). Communication-efficient and distributed learning over wireless networks: Principles and applications. *Proceedings of the IEEE*, *109*(5), 796–819.

Patel, V. M., Gopalan, R., Li, R., & Chellappa, R. (2015). Visual domain adaptation: A survey of recent advances. *IEEE Signal Processing Magazine*, *32*(3), 53–69.

Perez-Nieves, N., & Goodman, D. (2021). Sparse spiking gradient descent. *Conference on neural information processing systems (NeurIPS)*, *34*, 11795–11808.

ur Rehman, M. H., Liew, C. S., Abbas, A., Jayaraman, P. P., Wah, T. Y., & Khan, S. U. (2016). Big data reduction methods: A survey. *Data Science and Engineering*, *1*, 265–284.

Rubanova, Y., Chen, R. T. Q., & Duvenaud, D. K. (2019). Latent ODEs for irregularly-sampled time series. *Advances in Neural Information Processing Systems (NeurIPS)*, *32*.

Samadzadeh, A., Far, F. S. T., Javadi, A., Nickabadi, A., & Chehreghani, M. H. (2023). Convolutional spiking neural networks for spatio-temporal feature extraction. *Neural Processing Letters*, *55*(6), 6979–6995. https://doi.org/10.1007/s11063-023-11247-8

Schöne, M., Sushma, N. M., Zhuge, J., Mayr, C., Subramoney, A., & Kappel, D. (2024). Scalable event-by-event processing of neuromorphic sensory signals with deep state-space models. In *2024 International conference on neuromorphic systems (ICONS)* (pp. 124–131). IEEE.

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., & Benosman, R. (2018). Hats: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1731–1740).

Smith, J. T. H., Warrington, A., & Linderman, S. W. (2022). Simplified state space layers for sequence modeling. In *The Thirteenth International Conference on Learning Representations (ICLR)* 1–35

Stewart, K. M., Shea, T., Pacik-Nelson, N., Gallo, E., & Danielescu, A. (2023). Speech2spikes: Efficient audio encoding pipeline for real-time neuromorphic systems. In *Proceeding of the annual neuro-inspired computational elements conference (NICE)* (p. 71–78). https://doi.org/10.1145/3584954.3584995

Sun, Q., Liu, Y., Chua, T.-S., & Schiele, B. (2019). Meta-transfer learning for few-shot learning. In *Proceeding of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*.

Vincent-Lamarre, P., Calderini, M., & Thivierge, J.-P. (2020). Learning long temporal sequences in spiking networks by multiplexing neural oscillations. *Frontiers in Computational Neuroscience*, *14*. https://doi.org/10.3389/fncom.2020.00078

Wang, X., Wang, S., Ding, Y., Li, Y., Wu, W., Rong, Y., Kong, W., Huang, J., Li, S., & Yang, H. et al. (2024). State space model for new-generation network alternative to transformers: A survey.arXiv preprint arXiv:2404.09516.

Wiedemann, S., Mehari, T., Kepp, K., & Samek, W. (2020). Dithered backprop: A sparse and quantized backpropagation algorithm for more efficient deep neural network training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops* (pp. 720–721).

Xu, C., Chen, L., Chen, H., Chung, V., & Qu, Q. (2025). Towards end-to-end neuromorphic event-based 3d object reconstruction without physical priors. In *2025 IEEE International Conference on Multimedia and Expo (ICME)* 1–6

Yik, J., et al. (2025). The neurobench framework for benchmarking neuromorphic computing algorithms and systems. *Nature Communications*, *16*, 1545.

Yin, B., Corradi, F., & BohtÃ©, S. M. (2021). A surrogate gradient spiking baseline for speech command recognition. *Nature Machine Intelligence*, *3*. https://doi.org/10.1038/s42256-021-00397-w

Zhang, S., Yang, Q., Ma, C., Wu, J., Li, H., & Tan, K. C. (2024). Tc-lif: A two-compartment spiking neuron model for long-term sequential modelling. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(15), 16838–16847. https://doi.org/10.1609/aaai.v38i15.29625

Zubic, N., Gehrig, M., & Scaramuzza, D. (2024). State space models for event cameras. In *Proceeding of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 5819–5828).

Zucchet, N., & Orvieto, A. (2024). Recurrent neural networks: Vanishing and exploding gradients are not the end of the story. In *Advances in Neural Information Processing Systems (NeurIPS) 37* 139402–139443.

Zuckerman, L. P., Naor, E., Pisha, G., Bagon, S., & Irani, M. (2020). Across scales and across dimensions: Temporal super-resolution using deep internal learning. In *Proceeding of European conference on computer vision*.