ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exploring Multi-bit Spike Trains in Neuromorphic Computing

Bachelor Thesis

Yi Zhu

Wednesday 19th February, 2025

Advisors: Dr. G. Kwasniewski, P. Okanovic, Prof. Dr. T. Hoefler
Department of Computer Science, ETH Zürich

Abstract

Spiking Neural Networks (SNNs) represent an emerging paradigm in machine learning, inspired by biological neurons, with potential for greater energy efficiency compared to traditional Artificial Neural Networks (ANNs). Unlike ANNs, which use continuous floating-point outputs, SNNs produce discrete (binary) spikes when a neuron's membrane voltage exceeds a threshold.

In this thesis we propose a novel firing model where neurons fire at multiple spiking levels (multi-bit spike trains) when their membrane potentials reach corresponding thresholds. We show that this model enables up to 51% faster convergence and up to 2% better validation accuracy on various tasks compared to the traditional 1-bit spike train model, while maintaining the energy efficiency of SNNs. We also present an energy consumption model in a real-world workflow and show the multi-bit spike train model can be up to 60% more energy efficient than the 1-bit spike train model by reducing the number of time steps.

Contents

Chapter 1

Introduction

Artificial neural networks (ANNs) [1] have been widely used in machine learning and artificial intelligence. The computational principle behind ANNs is based primarily on matrix multiplications which can be efficiently implemented on modern hardware like GPUs [2]. Although such operations can be excellently parallelized and accelerated, the energy consumption of ANNs is still high.

In contrast, human brains are much more energy-efficient than ANNs. The energy consumption of a human brain is estimated to be around 20 watts [3], while the energy consumption of a large ANN model like ChatGPT is estimated to be around 23.5 megawatts [4]. As such, the human brain is estimated to be over one million times more energy-efficient than ChatGPT. One of the reasons for this energy efficiency is the difference between the neuron models used in ANNs and the biological neurons in the human brain. Unlike ANNs, which use floating-point numbers for numerical computation, biological neurons communicate with each other using spike trains and rely on temporal information [5].

To model such biological neurons and deploy them in artificial neural networks, spiking neural networks (SNNs) have been proposed. The most commonly used neuron model in SNNs is leaky integrate-and-fire (LIF) [6]. In this model, a neuron's membrane potential is increased by incoming spikes and decreased by a leak term. When the membrane potential reaches a threshold, the neuron fires a spike and resets its membrane potential. The communication between neurons in SNNs is done by sending spikes, which are binary events.

In this thesis, we propose a novel neuron model where neurons can fire at multiple spiking levels. Instead of firing a single spike when the membrane potential reaches a threshold, neurons can fire multiple spikes at different levels. We call this model the multi-bit spike train model. The motivation

behind this model is to increase the communication bandwidth between neurons and enable more efficient training or inference in SNNs. We explore the properties of this model and compare it with the traditional LIF neuron model.

We experiment on various datasets and tasks with the multi-bit spike train model and show that it can achieve better performance than the classic 1-bit spike train model in most cases. Finally, we also present an energy consumption model of the multi-bit spike train model and show that under the assumption of certain hardware implementations it can be more energy-efficient than the LIF neuron model.

Chapter 2

Background

2.1 Leaky Integrate-and-Fire Neuron Model

2.1.1 Dynamics of the Membrane Potential

A leaky integrate-and-fire (LIF) neuron [6] is a simple model of that captures the essential dynamics of a neuron. The LIF model is described by the differential equation:

$$\tau \frac{\mathrm{d}U(t)}{\mathrm{d}t} = -U(t) + I_{\text{in}}(t) \tag{2.1}$$

where $U(t)$ is the neuron's membrane potential, τ is its time constant, and $I_{\text{in}}(t)$ is the input current to the neuron. The neuron fires a spike when the membrane potential reaches a threshold U_{th}. The membrane potential is then reset to a resting potential U_{rest}.

The equation above has an approximate solution given by:

$$U[t] = \beta U[t-1] + (1-\beta) I_{\text{in}}[t] \tag{2.2}$$

where $\beta = e^{-\Delta t/\tau}$, Δt is the time step, and $I_{\text{in}}[t]$ is the input current at time t, defined by the following equation:

$$I_{\text{in}}[t] = W \cdot X[t] \tag{2.3}$$

where $X[t]$ is the input spike train at time t, and W is the weight matrix.

Since the weights W are learnable parameters, one often merges the weights with the coefficient $(1-\beta)$, thus obtaining the following equation:

$$U[t] = \beta U[t-1] + W \cdot X[t] - S_{\text{out}}[t] \cdot \theta \tag{2.4}$$

where $S_{\text{out}}[t]$ is the output spike train at time t, and θ is the threshold of the neuron. By subtracting $S_{\text{out}}[t] \cdot \theta$ from the equation, one resets the membrane potential (soft reset) when the neuron fires a spike.

2.1.2 Spiking Mechanism

The firing model, as defined by Lapique in 1907 [6], is modeled by the following equation:

$$S_{\text{out}}[t] = \begin{cases} 1 & \text{if } U[t] \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

This is a heaviside step function, which is not differentiable.

For the simplicity of analysis, one often considers $x := U[t] - \theta$ and $y := S_{\text{out}}[t]$. At each time step, the membrane potential $U[t]$ is evaluated through

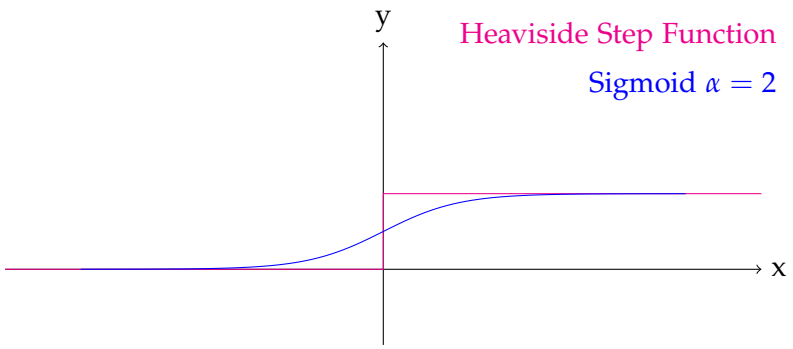


Figure 2.1: Comparison of the Heaviside Step Function and the Sigmoid Function

the Heaviside step function, and the output is a binary value of 0 or 1, as illustrated in Figure 2.1. The array of these binary values in the time domain is called the spike train of the neuron.

After the neuron fires a spike, the membrane potential is reset to the resting potential. There are two ways to reset the membrane potential: hard reset and soft reset. In the hard reset, the membrane potential is immediately set to the resting potential. In the soft reset, the membrane potential is subtracted by the threshold when the neuron fires a spike. Although the hard reset is more biologically plausible and more efficient in terms of computation, the soft reset is more popular in practice because it often delivers better performance in training [7]. In this thesis we focus on the soft reset.

2.2 Training of Spiking Neural Networks

2.2.1 Constructing Spiking Neural Networks

The construction of spiking neural networks is similar with the construction of traditional artificial neural networks. One of the most popular methods to construct a spiking neural network is to use the LIF neuron node to replace the ReLU activation function in the ANNs.

There are also other techniques to replace some components of the ANNs with certain tweaks for the SNNs, e.g. variants of batch normalization (such as Spike-Norm [8]) and attention mechanisms (such as Spiking RWKV [9]), but they are outside of the scope of this thesis.

2.2.2 Backpropagation through Time

Although our brains are likely not trained by backpropagation, the gradient-based optimization is still the most popular and reliable method to train the neural networks.

Backpropagation through time (BPTT) is a method used to train recurrent neural networks (RNNs) [10] by unfolding the network in time and applying backpropagation. The same method can be applied to train SNNs, as they are very similar to RNNs.

There are also other methods to train SNNs, like SLAYER [11] and EXODUS [12] which utilize the vectorized model of the SNNs. They introduce a new way to calculate the gradients that approximate the gradients calculated by BPTT. Since we are more interested in the performance of the multi-bit spike train model in convergence and firing rate, we will focus on the temporal model of the SNNs in this thesis.

2.2.3 Surrogate Gradient

Gradient-based optimization requires the activation function to be differentiable, which the Heaviside step function is not. Therefore, one often uses another differentiable function to approximate the Heaviside step function in the backpropagation algorithm [13], e.g. the sigmoid function (see Figure 2.1):

$$S'_{\text{out}}[t] = \frac{1}{1 + e^{-\alpha \cdot x}} \tag{2.6}$$

It turns out that SNNs can tolerate such approximations, and it has become the state-of-the-art method to train performant SNNs.

Chapter 3

Multi-bit Spike Train Model

3.1 Graded Spikes

We now introduce our novel firing model with graded spikes, namely our multi-bit spike train model. We divide the range of $[0, \theta]$ into $2^n - 1$ intervals where n is the number of bits used to encode one single spike. Then once the membrane potential reaches the threshold for a certain interval, the neuron fires a spike with the corresponding intensity described as follows:

$$S_{\text{out}}[t] = \begin{cases} 0 & \text{if } U[t] < \frac{1}{2^n-1} \cdot \theta \\ \frac{i}{1^n-1} & \text{if } \frac{i}{2^n-1} \cdot \theta \leq U[t] < \frac{i+1}{2^n-1} \cdot \theta,\ i \in [1, 2^n - 2] \\ 1 & \text{if } U[t] \geq \theta \end{cases} \tag{3.1}$$

When $n = 1$, the multi-bit spike train model becomes the binary spike train model.
Again we focus on the case $x := U[t] - \theta$ and $y := S_{\text{out}}[t]$, the spike function is now a step function with $2^n - 1$ steps, illustrated in Figure 3.1.

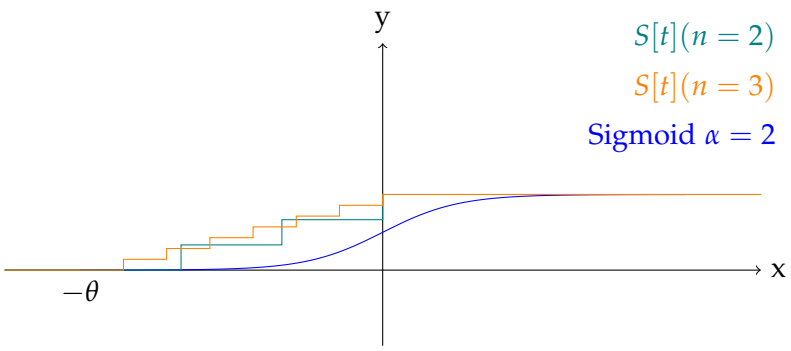


Figure 3.1: Comparison of the Multi-bit Spike Train Model and the Sigmoid Function

Here we use the linear span of the interval to represent the intensity of the

spike, because surrogate functions like sigmoid and atan are almost linear with the input value close to zero, which is the case when the threshold is set to a small value.

Intuitively this firing model enables higher information bandwidth between neurons. Although biologically, neurons can only fire binary encoded spikes, the spike trains can be rate encoding [14] or temporal encoding [15]. The rate encoding is based on the number of spikes in a certain time window, i.e. the neuron fires more spikes when the input is stronger. The temporal encoding is based on the timing of the spikes, i.e. the neuron fires earlier when the input is stronger. The multi-bit spike train model is a generalization where the graded spikes can be interpreted as rate encoding and their temporal positions can be interpreted as temporal encoding.

3.2 Shifted Surrogate Function

We now present a shifted sigmoid function which essentially serves as a surrogate function for the gradient calculation in backpropagation.

The multi-bit spike train model leads to the problem that our chosen surrogate function no longer approximates the spike function well. One can tell from the gap between the multi-bit spike train model and the sigmoid function in Figure 3.1 that the gradients derived from the sigmoid function can no longer reflex the gradients of the multi-bit spike train model accurately. A simple solution is to shift the sigmoid function accordingly. The following equation describes the shifted sigmoid function:

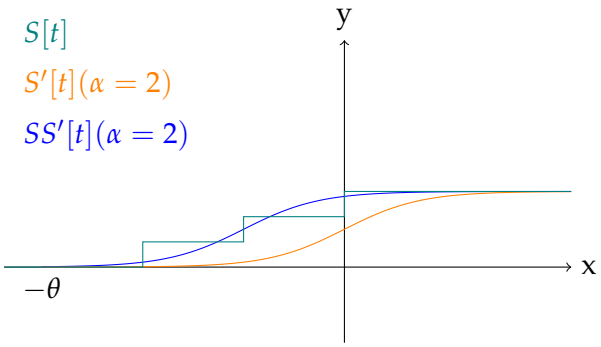
$$S'_{\text{out}}[t] = \frac{1}{1 + \exp(-\alpha \cdot (x + \frac{2^{n-1}-1}{2^n-1} \cdot \theta))} \tag{3.2}$$

This technique centers the sigmoid function relatively to the multi-bit spike train model, illustrated in Figure 3.2. It improves the accuracy of the resulting networks.

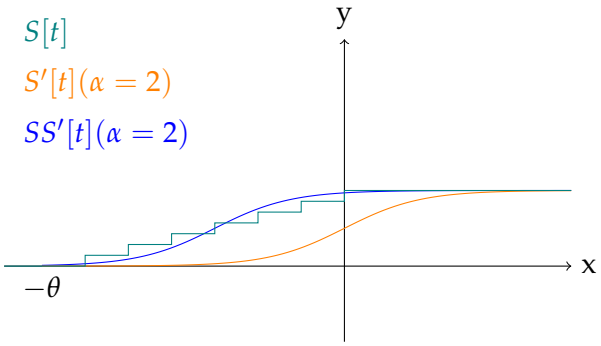
3.3 Implementation

We use the PyTorch based SNN framework SpikingJelly [16] to implement the multi-bit spike train model. In SpikingJelly, the firing mechanism is implemented as a forward path of the surrogate function. One just needs to override the Heaviside step function with $S_{\text{out}}[t]$ in Equation 3.1. The core implementation is shown in Figure 3.3.

Noticing that the input x is already centered to zero by SpikingJelly, we apply the shift directly to the sigmoid function.



(a) $n = 2$



(b) $n = 3$

Figure 3.2: Comparison of Sigmoid Function, Shifted Sigmoid Function and Multi-bit Spike Train Model: $S[t]$ is the output spikes of the multi-bit spike train model, $S'[t]$ is the output spikes of the sigmoid function, $SS'[t]$ is the output spikes of the shifted sigmoid function

```
@torch.jit.script
def multi_level(x: torch.Tensor, n: int, threshold: float):
    l = int(2**n)-1
    r = (x >= 0).float()
    for i in range(1, l):
        r += ((x >= -float(i)/l * threshold) ^ (x >= -float(i-1)/l * threshold)) * float(l-i)/l
    return r.to(x)

class sigmoid(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, alpha, n, threshold):
        shift = (2**(n-1) -1) / (2**n-1) * threshold
        if x.requires_grad:
            ctx.save_for_backward(x+shift)
            ctx.alpha = alpha
            ctx.n = n
            ctx.threshold = threshold
        return multi_level(x, n, threshold)

    @staticmethod
    def backward(ctx, grad_output):
        return sigmoid_backward(grad_output, ctx.saved_tensors[0], ctx.alpha, ctx.n, ctx.threshold)

class Sigmoid(SurrogateFunctionBase):
    def __init__(self, alpha=4.0, spiking=True, n=1, threshold=1.0):
        super().__init__(alpha, spiking, n, threshold)

    @staticmethod
    def spiking_function(x, alpha, n, threshold):
        return sigmoid.apply(x, alpha, n, threshold)

    @staticmethod
    def backward(grad_output, x, alpha, n, threshold):
        shift = (2**(n-1) -1) / (2**n-1) * threshold
        return sigmoid_backward(grad_output, x+shift, alpha, n, threshold)[0]
```

Figure 3.3: Implementation of the Multi-bit Spike Train Model in SpikingJelly

Chapter 4

Evaluation

4.1 Evaluation Aspects

In this chapter, we evaluate the multi-bit spike train model on various aspects:

- **Accuracy**: We compare the convergence speed and final accuracy of the multi-bit spike train model with the traditional 1-bit spike train model on various tasks and datasets.
- **Firing Rate**: We investigate how the increase of the bit width of the spike train affects the firing rate of the neurons in the network.
- **Quantizability**: We evaluate the accuracy of the multi-bit spike train model when quantized to lower precision.
- **Energy Consumption**: We evaluate the energy consumption of the multi-bit spike train model in practical scenarios.
- **Performance**: We present the performance data of the multi-bit spike train model on GPUs.

4.2 Experimental Setup

We implement the multi-bit spike train model with SpikingJelly [16] (version 0.0.0.0.15) using its PyTorch [17] (version 2.5.0) backend. All experiments are conducted in the NVIDIA PyTorch container (nvcr.io/nvidia/pytorch:24.08-py3) on an NVIDIA GH200 super chip unless otherwise stated.

The multi-bit spike train model and the traditional 1-bit spike train model are tested on various tasks and datasets. The regular static datasets are imported from Torchvision [18] while the neuromorphic datasets are imported from Tonic [19].

We measure the training accuracy (per batch), test accuracy (per epoch) and firing rate at certain points in the networks of the 1-bit to 8-bit spike train models. We then investigate the convergence, accuracy, firing rate, and quantizability of the multi-bit spike train model.

We mainly use the Fashion MNIST dataset [20] and a convolutional neural network (CNN) [21] to set up the experiments of an image classification task. We will also show the results also hold on more complex datasets like CIFAR-10 [22].

In the following sections, the input images from Fashion MNIST are converted to spike trains using the Poisson encoding method. The CNN has a structure of $28 \times 28 - 8c5 - 2a - 16c5 - 2a - 10o$, visually illustrated in Figure 4.1.

Unless stated otherwise, the experiments with Fashion MNIST are run with a batch size of 128, a learning rate of $2 \cdot 10^{-3}$, time steps of 10, Adam as the optimizer and a total of 5 epochs.

Each dataset mentioned in this thesis is paired with a specific model structure. The model structures are shown in Table 4.1, and they are never changed throughout the experiments.

Dataset	Model Structure
Fashion MNIST	$28 \times 28 - 8c5 - 2a - 16c5 - 2a - 10o$
MNIST	$28 \times 28 - 8c5 - 2a - 16c5 - 2a - 10o$
NMNIST	$34 \times 34 - 8c5 - 2a - 16c5 - 2a - 10o$
DVS Gesture	$2 \times 34 \times 34 - 8c5 - 2a - 16c5 - 2a - 10o$
CIFAR-10	$3 \times 32 \times 32 - 256c3 - 256c3 - 256c3 - 2a - 256c3 - 256c3 - 256c3 - 2a - 2048fc - 100fc - 10a - 10o$

Table 4.1: The model structures used in the experiments with different datasets

4.3 Convergence & Accuracy

The first noticeable difference between the multi-bit spike train model and the 1-bit spike train model is the convergence speed (see Figure 4.2). Even just by increasing the bit width of the spike train from 1 to 2, the convergence speed of the network is significantly improved.

The improvement in convergence speed can lead to a significant reduction in training time when considering a fixed target accuracy. This is especially noteworthy when considering diminishing returns of accuracy in regards to training time.

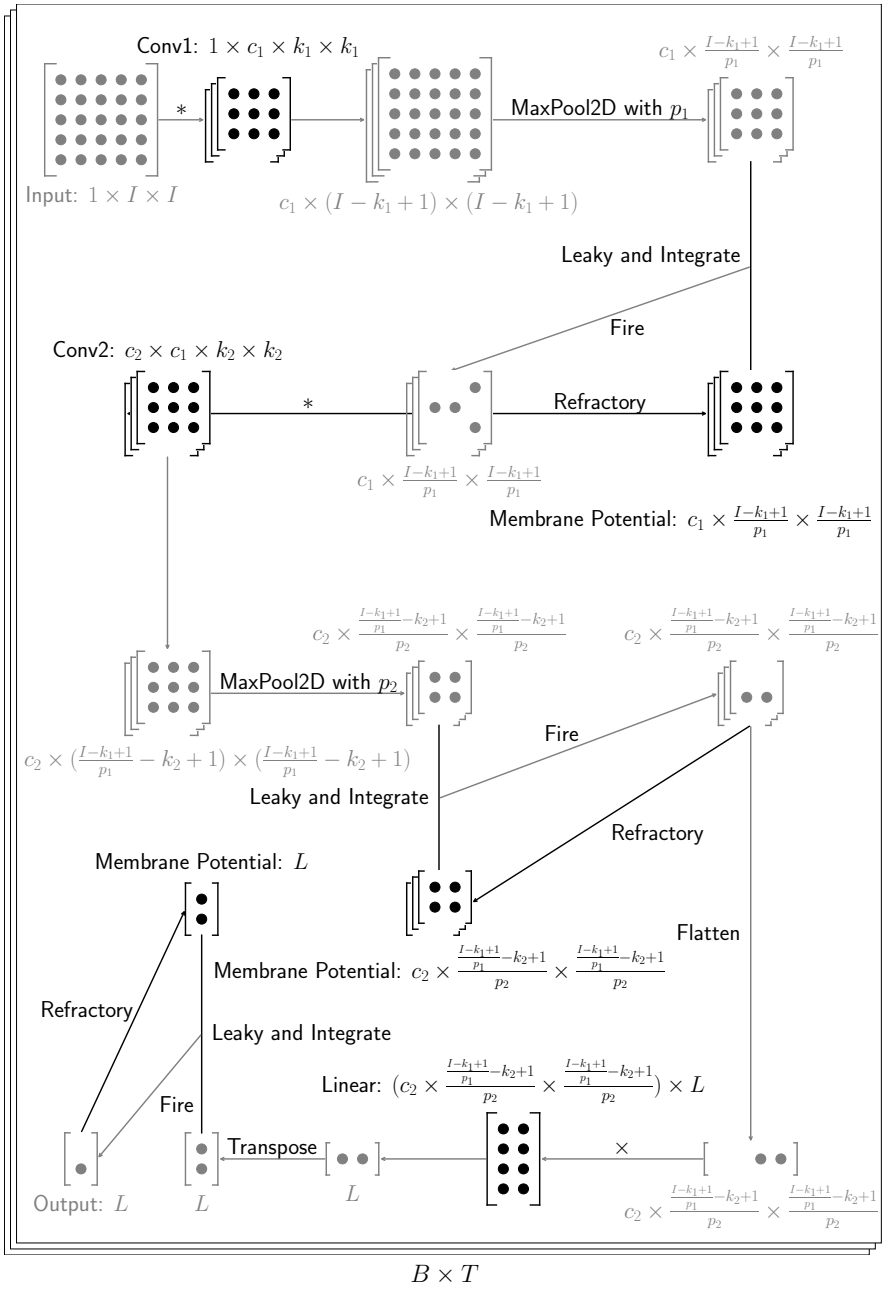
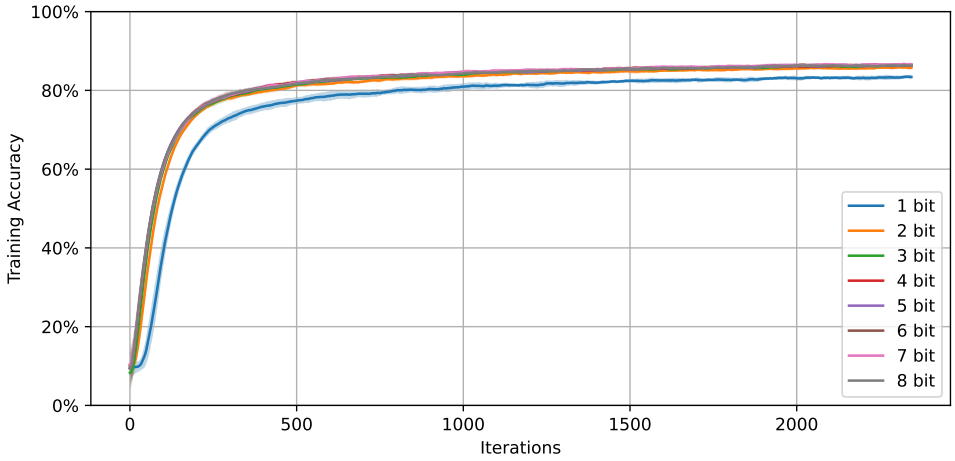
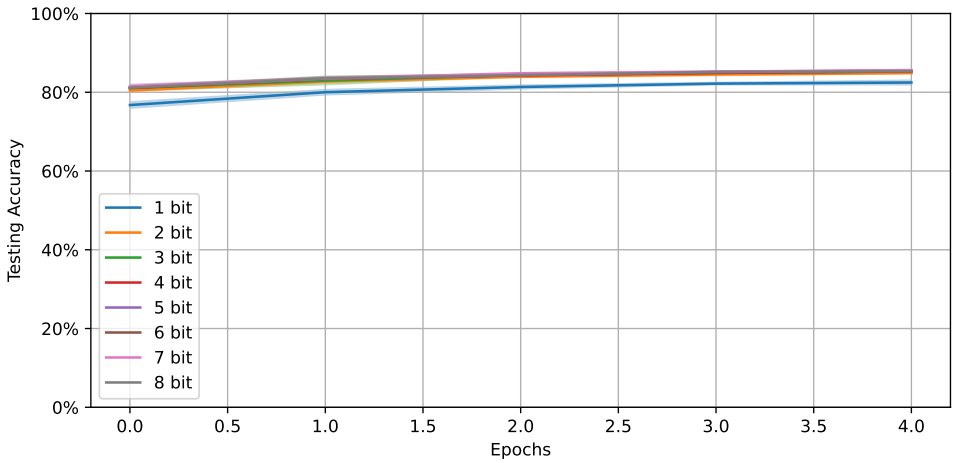


Figure 4.1: The CNN architecture used in the experiments with Fashion MNIST dataset



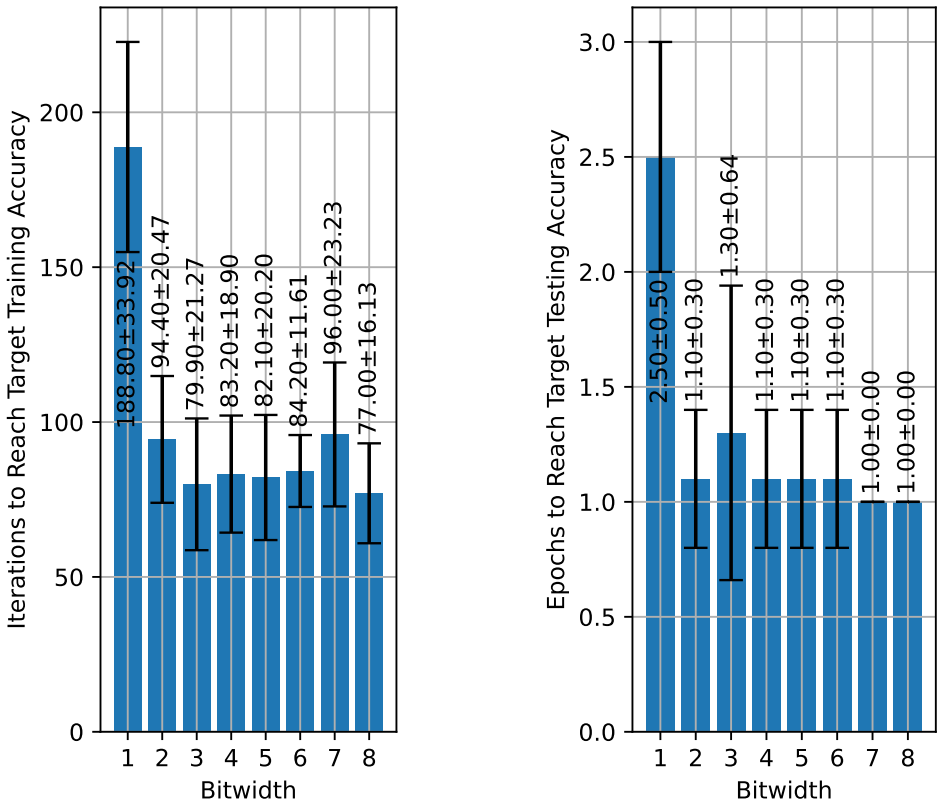
(a) Training Accuracy (smoothed with a window size of 100)



(b) Test Accuracy

Figure 4.2: Comparison of the Convergence Speed of 1-bit to 8-bit Spike Train Model, conducted with 10 different random seeds

Here we set the target accuracy to be 80%. The training time of the multi-bit spike train model is reduced by around 50%, shown in Figure 4.3.



(a) Iterations to reach 80% training accuracy **(b)** Epochs to reach 80% testing accuracy

Figure 4.3: Comparison of the Training Time of 1-bit to 8-bit Spike Train Model, conducted with 10 different random seeds

This improvement however also comes at certain cost:

- The improvement in convergence speed diminishes as the bit width of the spike train increases. The improvement from 2-bit to 3-bit ($\sim$ 15.4%) or even higher bit width (at most $\sim$ 59.2% reduction in iterations compared to the 1-bit baseline) is not as significant as the improvement from 1-bit to 2-bit ($\sim$ 50% reduction in iterations).
- The more complicated the model is, the harder is it to train the multi-bit spike train model. One can easily run into the problem of overfitting, for example in the case of DVS gesture recognition task A.1.

Due to the faster convergence speed, the multi-bit spike train model can achieve better accuracy than the 1-bit spike train model given a fixed number

of iterations or epochs for the most cases (see Figure 4.4).

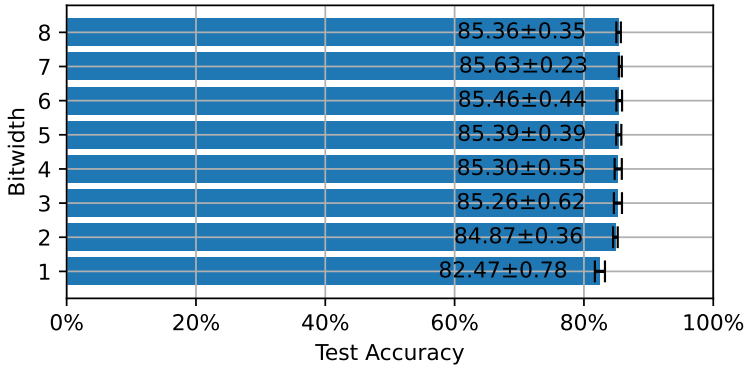


Figure 4.4: Comparison of the Final Accuracy of 1-bit to 8-bit Spike Train Model after 5 epochs, conducted with 10 different random seeds

In general, such behaviors can be shown on MNIST [23], NMNIST [24], Fashion MNIST, DVS gesture recognition [25], and CIFAR-10 datasets (see Appendix A).

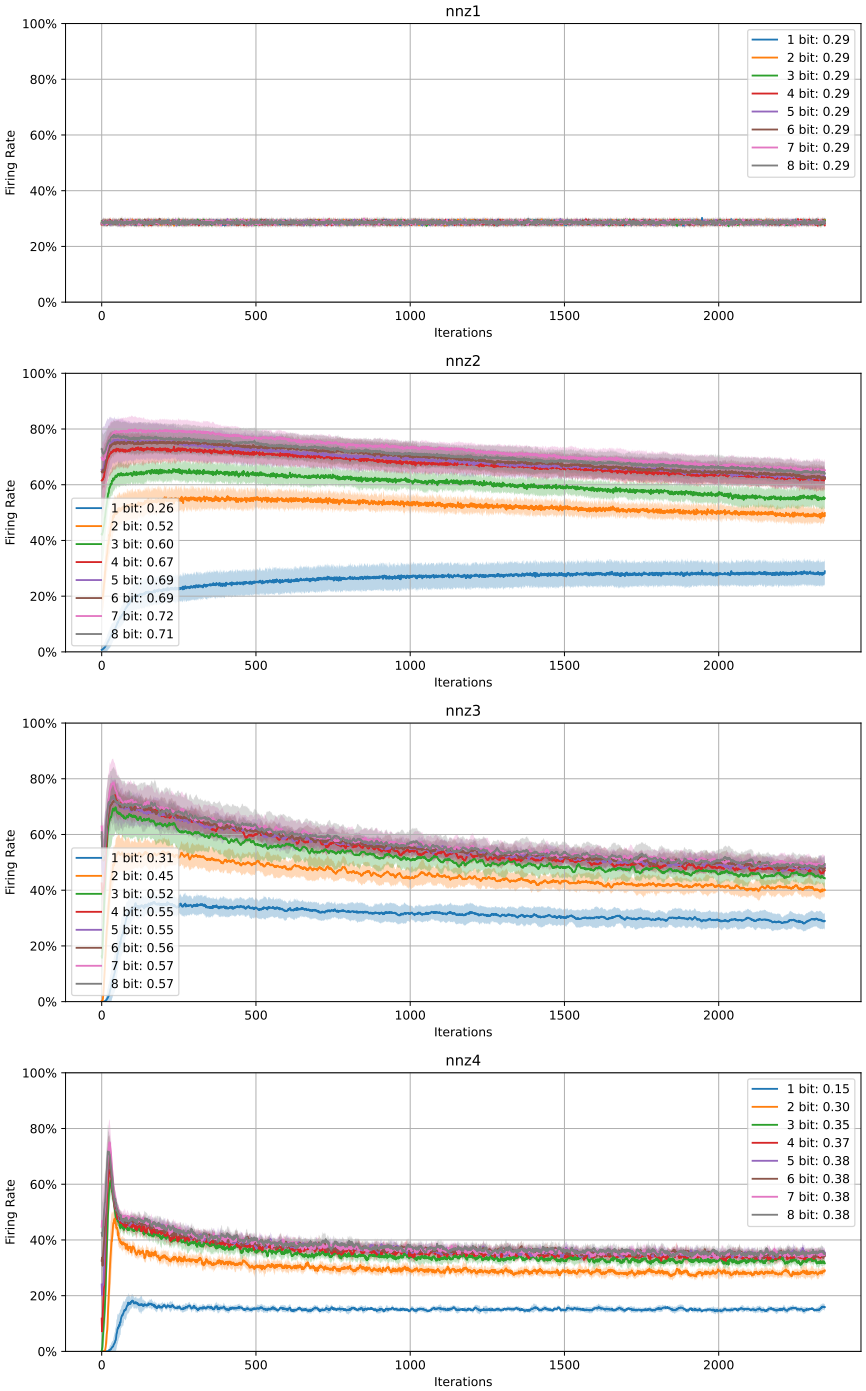
4.4 Firing Rate

SNNs are known for their sparsity in the firing rate of the neurons. We measure the firing rate of the LIF neuron layers at different positions in the models. Let "nnz[i]" denote the number of non-zero elements in the *i*-th measuring point, corresponding to the position from the inputs to the output layer. Measurement point 1 is always the input data, and the last measurement point is always the output data. Here we notice that the multi-bit spike train model has a higher firing rate than the 1-bit spike train model (see Figure 4.5), as tradeoff for the improvement in convergence speed.

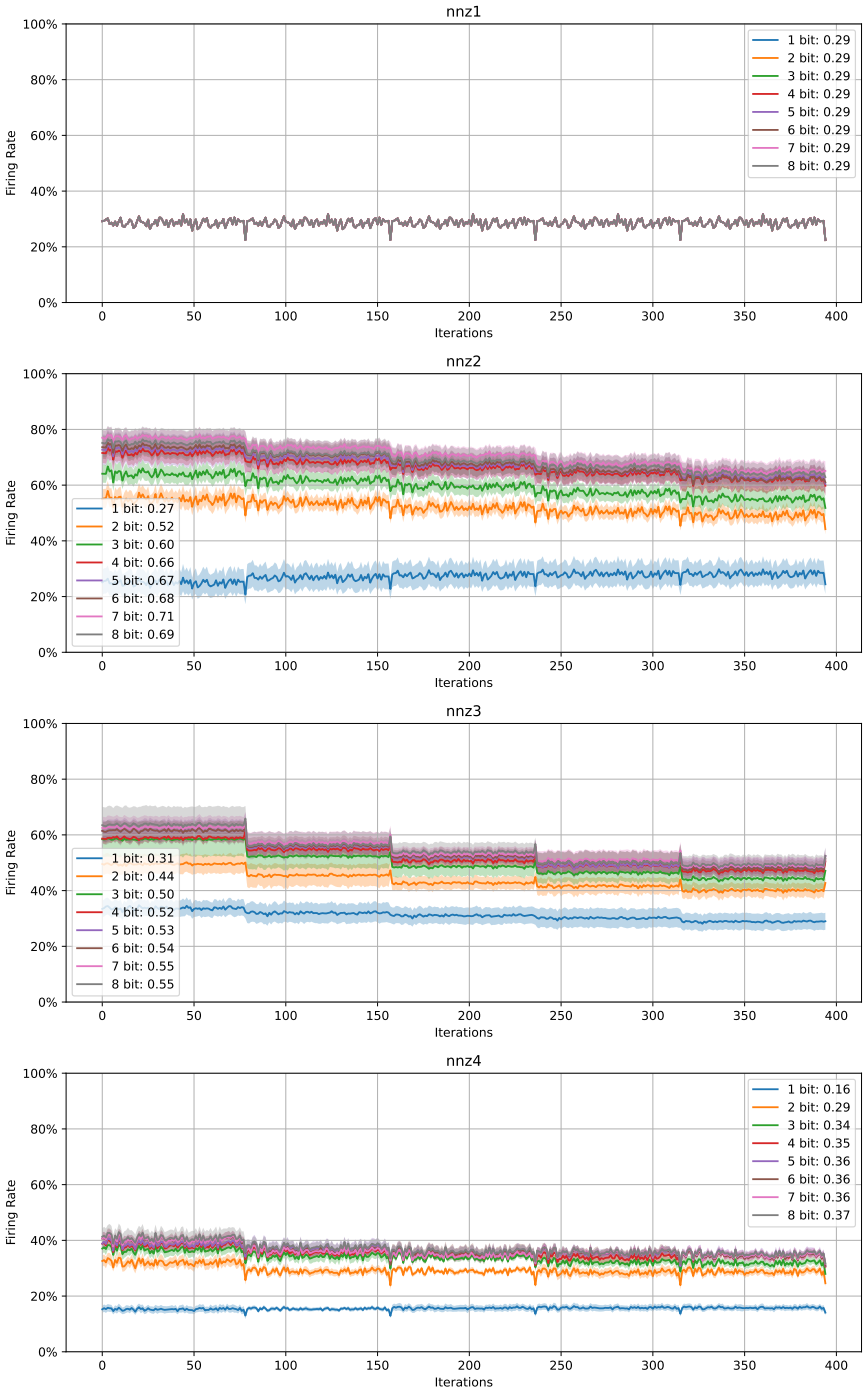
One can notice that the firing rate of the multi-bit spike train model peaks at the very beginning of the training session, when the model converges rapidly. The firing rate then decreases as the training progresses. If we extend the training session, the firing rate of the multi-bit spike train model tends to converge to the firing rate of the 1-bit spike train model. However, such process takes however a long time (see Figure 4.6).

Towards the end of the training time, the firing rate of the multi-bit spike train model is often still higher than the 1-bit spike train model, but sometimes comparable to the 1-bit spike train model (see Figure 4.7).

Such behavior is more visible on simpler datasets like MNIST. For now, we do not have any explanation for this phenomenon (see Appendix B).

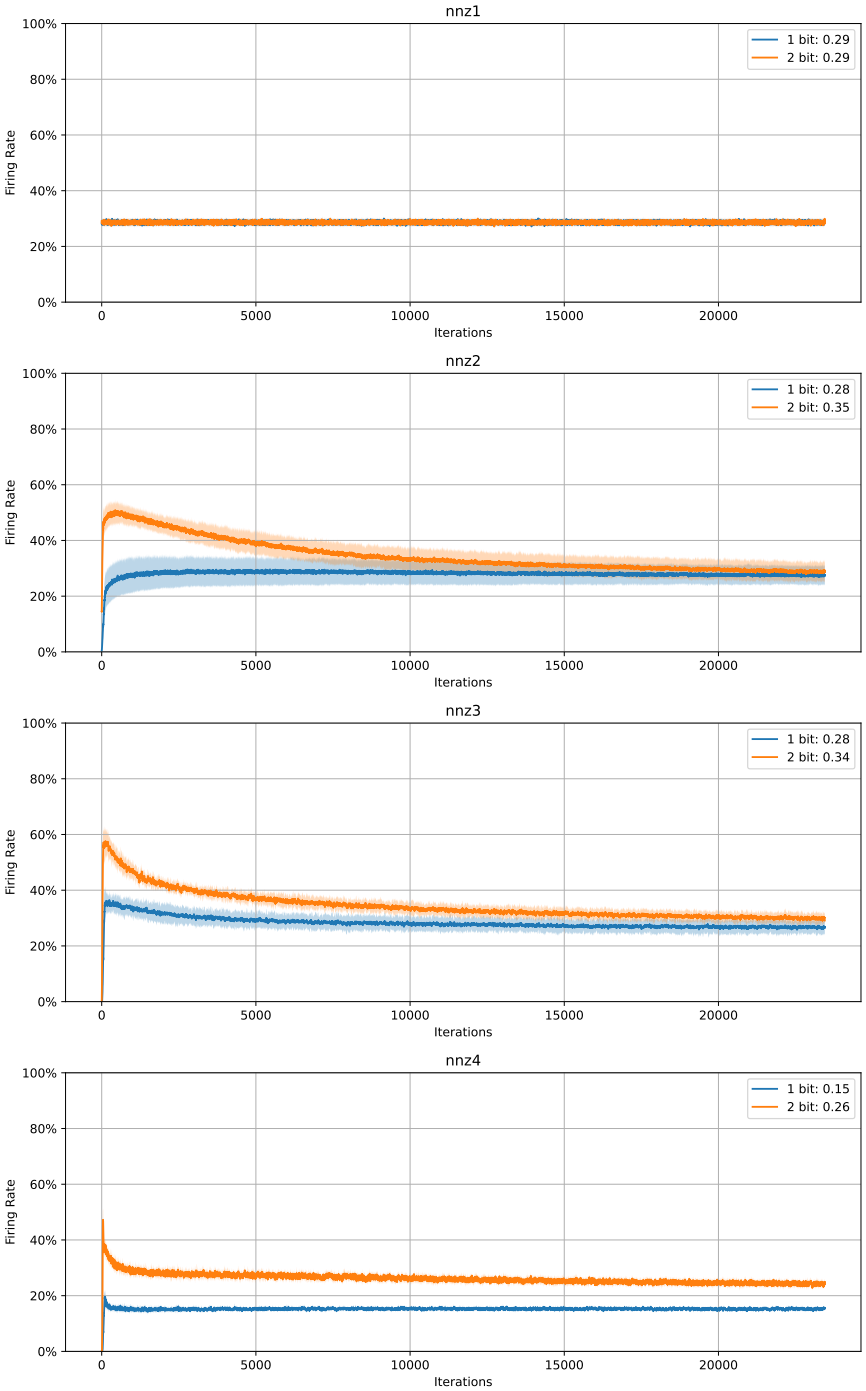


(a) Training Firing Rate

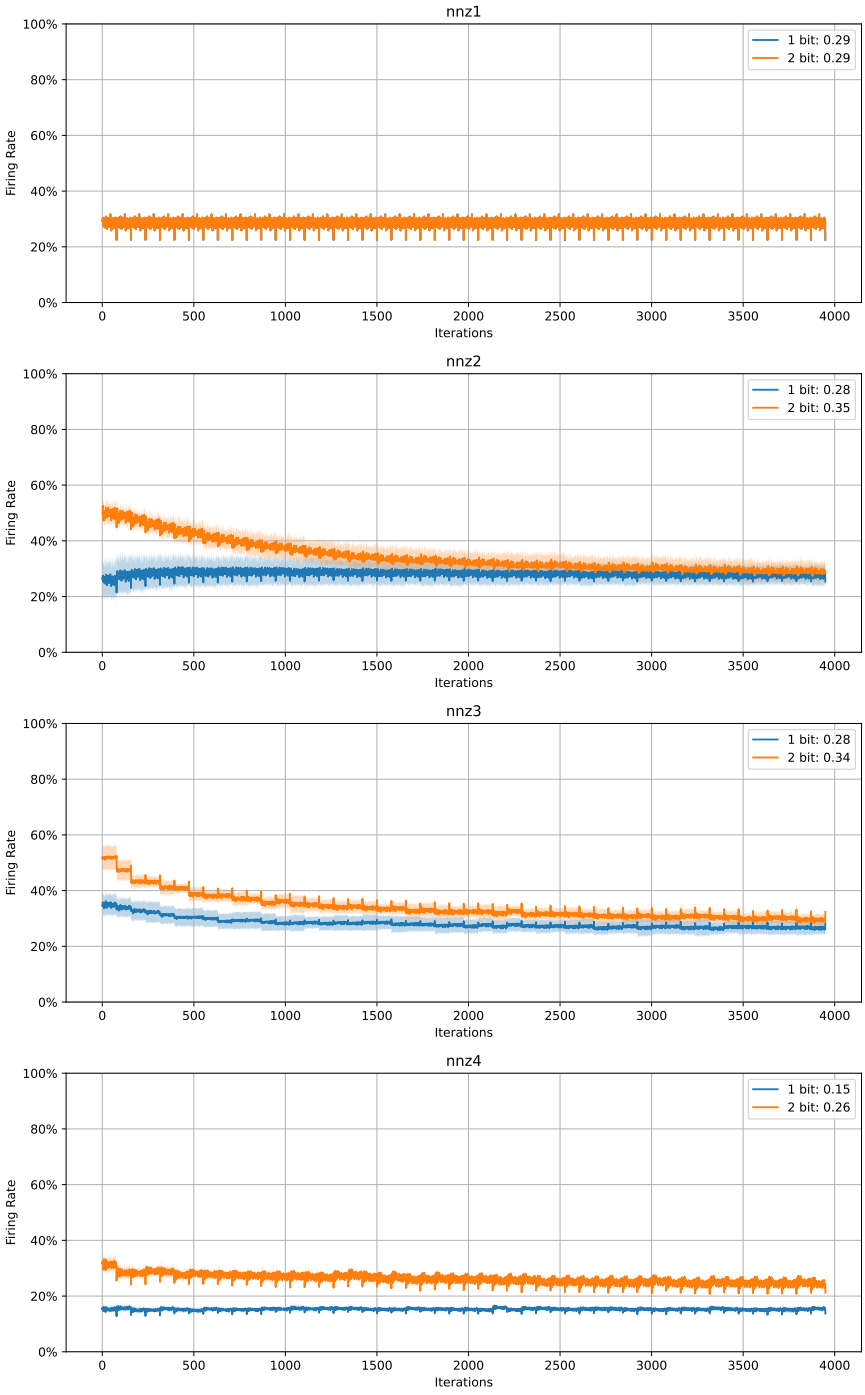


(b) Test Firing Rate

Figure 4.5: Comparison of the Firing Rate of 1-bit to 8-bit Spike Train Model, conducted with 10 different random seeds



(a) Training Firing Rate



(b) Test Firing Rate

Figure 4.6: Analog to figure 4.5, but with a training session of 50 epochs instead of 5 epochs, only the 1-bit and 2-bit spike train models

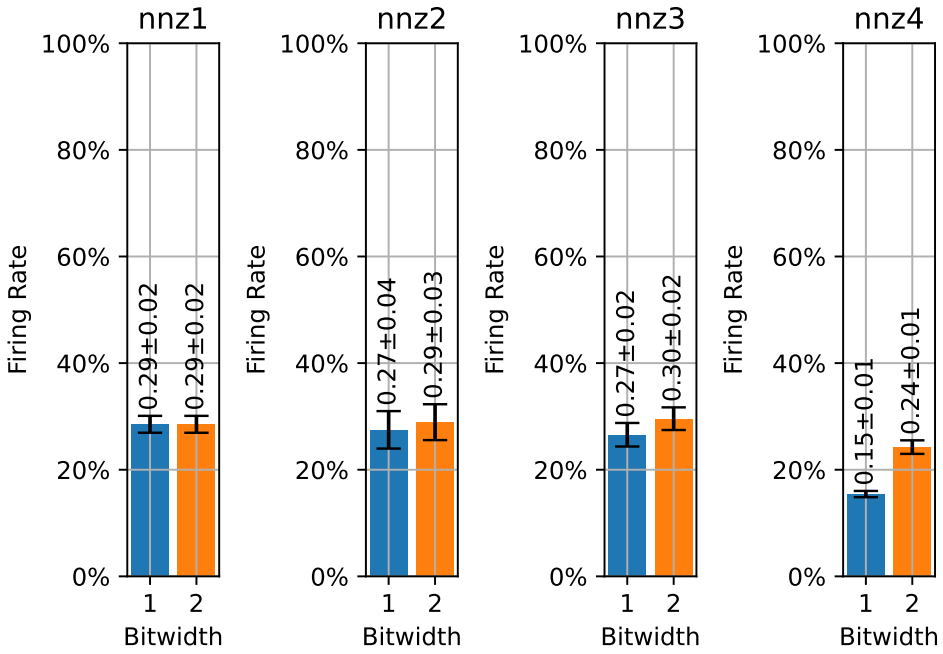


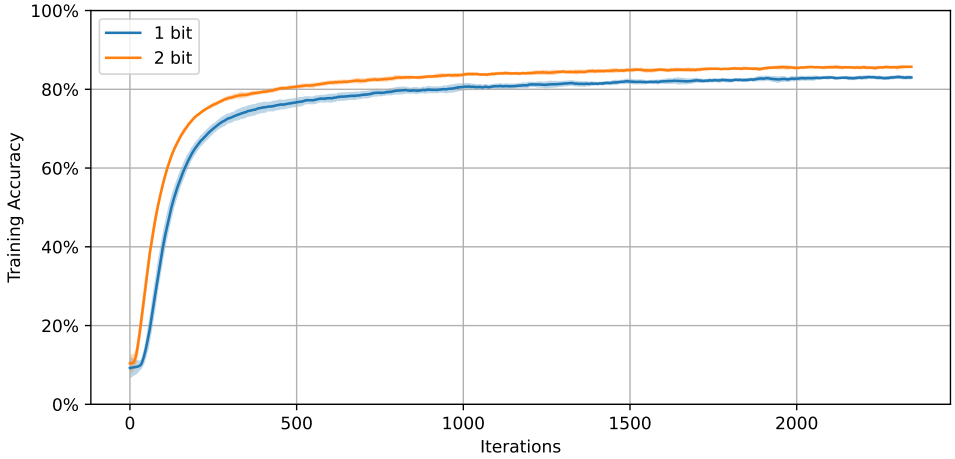
Figure 4.7: Comparison of the Final Firing Rate of 1-bit to 8-bit Spike Train Model after 50 epochs, only the 1-bit and 2-bit spike train models

4.5 Quantizability

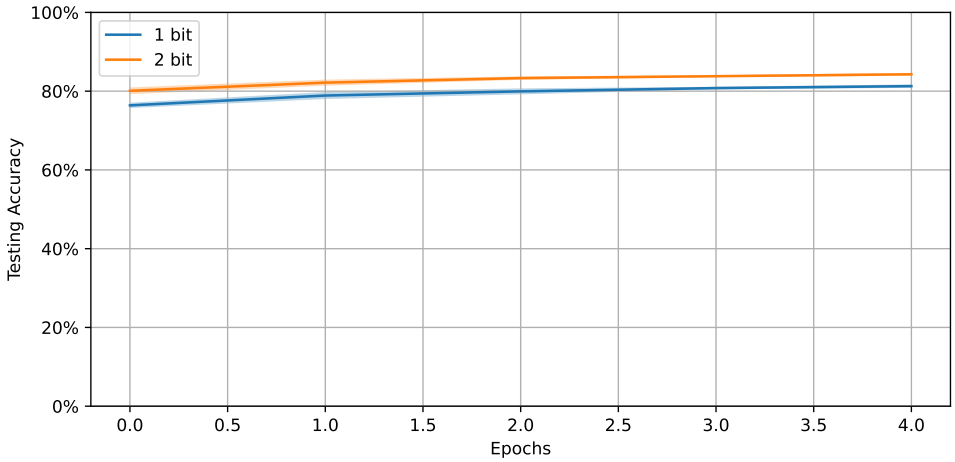
A common technique to reduce the memory footprint of SNNs without having significant impact on accuracy is quantization [26]. Quantization is the state-of-the-art technique to reduce the model size by using lower precision for parameters like weights and biases. There are studies showing that the weights and activations of SNNs can be quantized to 1-bit or 2-bit without significant loss in accuracy [27]. Here we show that both the multi-bit spike train model and the 1-bit spike train model can be trained with `bf16` and quantized to `int8` without significant loss in accuracy.

As shown in Figure 4.8, both SNN models converge well as if they were trained with `float32`. Meanwhile the multi-bit spike train model has its advantage in the convergence speed and accuracy preserved.

Before quantizing the models to `int8`, we first apply quantization-aware training to both SNN models. While regular ANNs may encounter various problems due to the high sensitivity in inputs and parameters, both SNN models here converge well. We also see that the multi-bit spike train model has a faster convergence speed and better accuracy than the 1-bit spike train model (Figure 4.9).



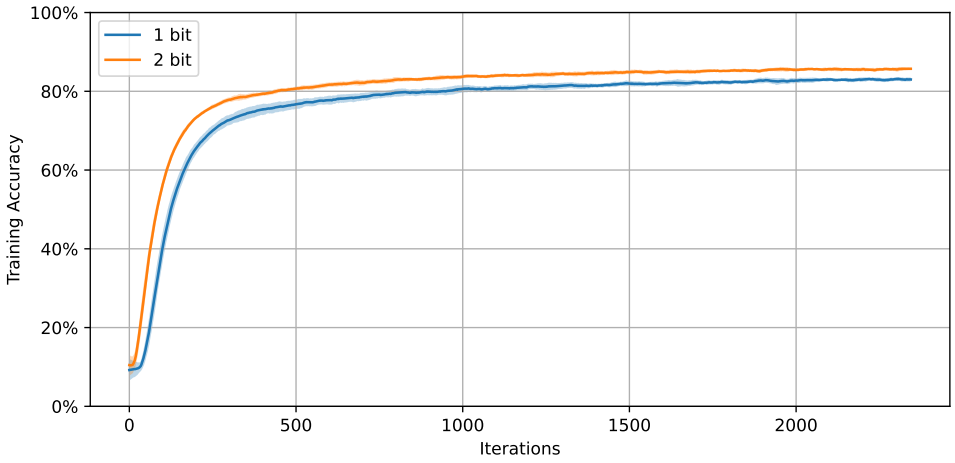
(a) Training Accuracy with half precision (smoothed with a window size of 100)



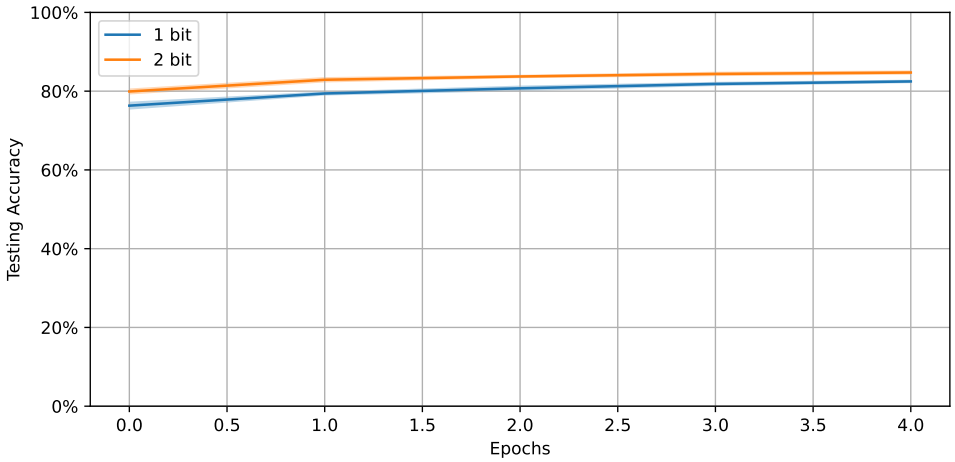
(b) Test Accuracy with half precision

(c) Final Accuracy with half precision after 5 epochs

Figure 4.8: Comparison of the Convergence Speed of 1-bit to 8-bit Spike Train Model with half precision, conducted with 10 different random seeds



(a) Training Accuracy with quantization-aware training (smoothed with a window size of 100)



(b) Test Accuracy with quantization-aware training

(c) Final Accuracy with quantization-aware training after 5 epochs

Figure 4.9: Comparison of the Convergence Speed of 1-bit to 8-bit Spike Train Model with quantization-aware training, conducted with 10 different random seeds

We then quantize the weights and biases to `int8` using the PyTorch quantization API. The quantization of the LIF layer is not supported at the moment. We evaluate the quantized models on the test set. The final accuracy is barely affected by the quantization (see Figure 4.9c). In comparison to the results in Figure 4.4, the final accuracy of the quantized 2-bit spike train model is dropped by around 0.59% while reducing the memory footprint by almost 75%.

4.6 Energy Consumption

One of the main motivations for SNNs is their energy efficiency compared to ANNs on specialized hardware like neuromorphic chips. Products like Loihi from Intel [28] and TrueNorth from IBM [29] have proved the potential of SNNs by utilizing the asynchronous communication via spikes. In the real-world scenario, one often uses accelerators like GPUs to train SNNs and deploy them on specialized hardware to achieve fast training and energy efficent inference.

Here we present an energy consumption model for the multi-bit spike train model and compare it with the 1-bit spike train model. We consider the unique properties of various hardware implementations and give the energy consumption of the multi-bit spike train model relative to the 1-bit spike train model.

4.6.1 Training Energy Consumption on GPUs

Popular SNN frameworks like snnTorch and SpikingJelly do not utilize the low precision of spikes and the sparsity of spike trains. The intermediate results are in practice dense 32-bit matrices. So in this case, the firing rate and the bit width of the spike train do not affect the energy consumption of the training phase.

The only factors that matter are the number of iterations required to reach a certain accuracy and the number of time steps that the network is simulated, assuming fixed network topology and batch size.

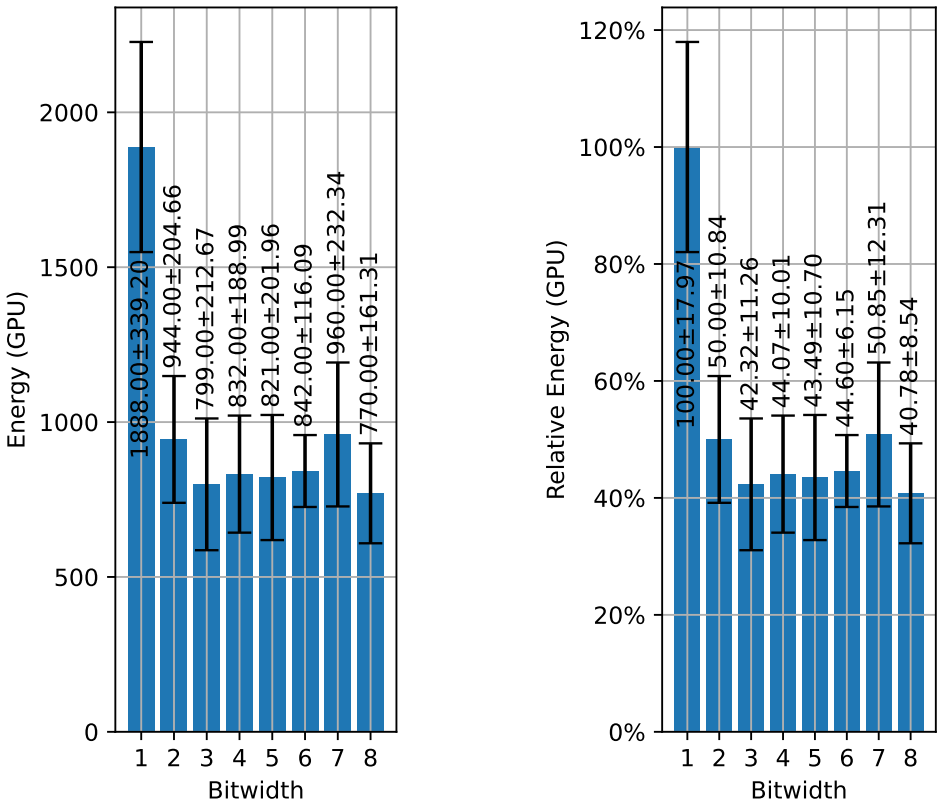
This allows us to create a simple, yet effective energy consumption model for the training phase on the GPUs. Let T_i denote the number of time steps, S_i denote the number of iterations required to reach a certain accuracy, and $E_{\text{train}-i}$ denote the energy consumption. Then we have:

$$E_{\text{train}-i} = T_i \cdot S_i \cdot c \tag{4.1}$$

where c is a constant factor that depends on the hardware and the software used, yet remains the same across different bit widths of the spike train.

As noticed in Section 4.3, one requires fewer iterations to reach a certain accuracy with the multi-bit spike train model. Here we focus on the energy consumption of the 2-bit spike train model, as it does not increase the firing rate as much as the other higher bit width models while still providing a significant improvement in convergence speed and accuracy.

Based on the results in Section 4.3, we can estimate the energy consumption of the 2-bit spike train model relative to the 1-bit spike train model directly by comparing the number of iterations required to reach a certain accuracy which is around $50.00 \pm 10.84\%$ in this case (see Figure 4.10).



(a) Energy Consumption Estimation, Unit in Constant c

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure 4.10: Training Energy Consumption Estimation on GPUs for Fashion MNIST Dataset

More details with other datasets (MNIST, NMNIST, DVS Gesture and CIFAR10) can be found in Appendix C.1.

4.6.2 Inference Energy Consumption on Neuromorphic Chips

A widely adopted (e.g. in [9] and [30]) energy estimation model for neuromorphic chips is the following:

$$E_{\text{inference}} \approx F \cdot fr \cdot E_{\text{AC}} \cdot T \tag{4.2}$$

where F is the number of floating point operations required to simulate the network, fr is the firing rate of the input neurons, E_{AC} is the energy consumption of accumulation, and T is the number of time steps.

Despite this, it is difficult to evaluate the energy consumption of the multi-bit spike train model on neuromorphic chips like Loihi and TrueNorth, as they do not support multi-bit spikes natively. Although it may be possible to encode the multi-bit spikes into multiple spikes with different intensities, the energy consumption of such encoding would be very expensive due to the high cost of the synchronization barrier between the time steps.

A viable option would be to consider hardware like Intel Loihi 2 which supports graded spikes up to 32-bit precision. We consider the case of Intel Loihi 2 and make the following assumptions:

1. There is no difference in the energy consumption for the number of bits used to encode the spikes.
2. The neuromorphic hardware performs add-and-accumulate operations whenever a spike is received.

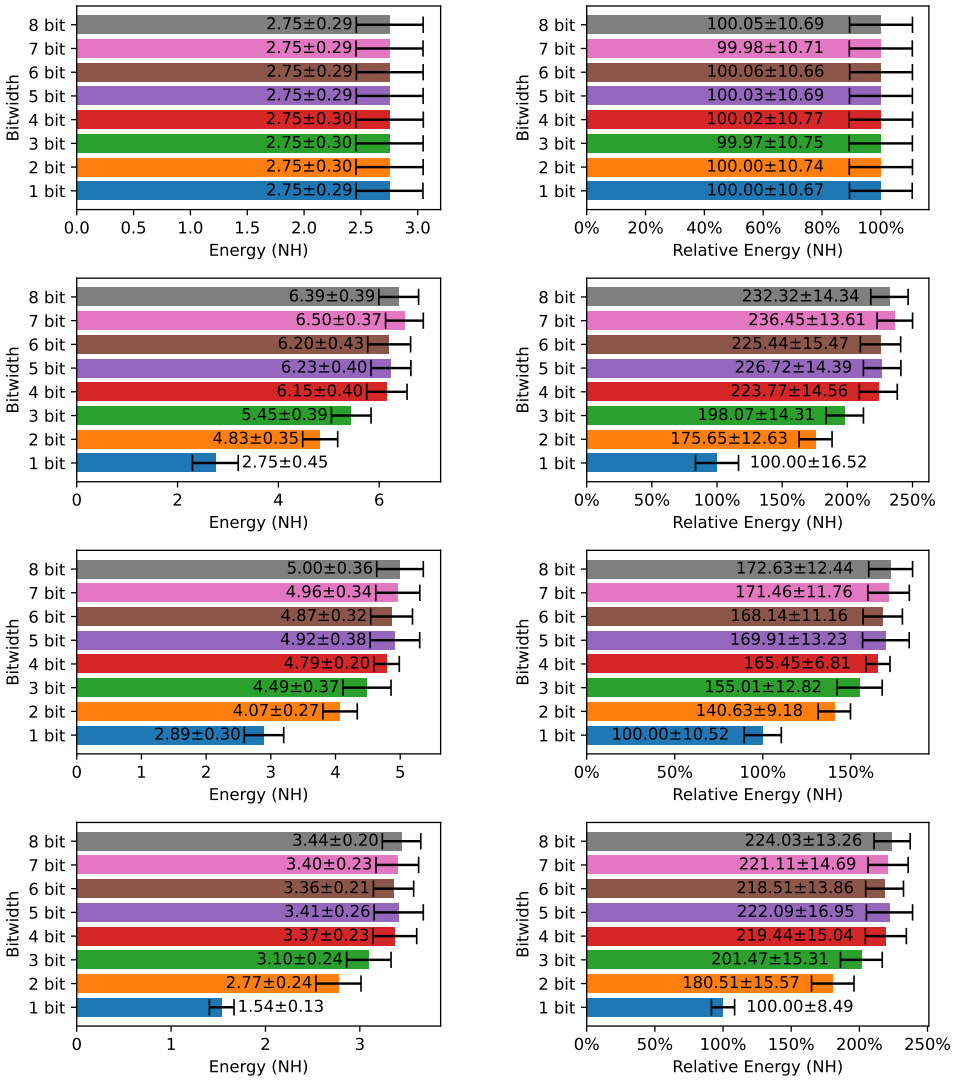
Since the payload is not variable in this case, the energy consumption of the multi-bit spike train model is directly proportional to the firing rate given a fixed maximum number of floating point operations and time steps. Analogously to the equation 4.2, we have the following estimation for the i-bit spike train model:

$$E_{\text{inference}-i} \approx F \cdot fr_i \cdot E_{\text{AC}} \cdot T_i \tag{4.3}$$

We can estimate the energy consumption of the 2-bit spike train model relative to the 1-bit spike train model by comparing the firing rate of the neurons. As expected, the energy consumption of the multi-bit spike train model is higher than the 1-bit spike train model, as the firing rate of the neurons is higher (see Figure 4.11).

More details with other datasets (MNIST, NMNIST, DVS Gesture and CIFAR10) can be found in Appendix C.2.

We consider $E_{\text{AC}} = 0.9\text{pJ}$, $E_{\text{MAC}} = 4.5\text{pJ}$ (energy cost for multiply-and-accumulate operations) [31] to estimate the precise energy consumption of the multi-bit spike train model and the ANN for Fashion MNIST dataset (see Figure 4.12). Due to the small network size and the high firing rate, the energy consumption of the multi-bit spike train model is higher than the ANN in some cases.



(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{AC}}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure 4.11: Inference Energy Consumption Estimation on Intel Loihi 2 for Fashion MNIST Dataset, conducted with 10 different random seeds

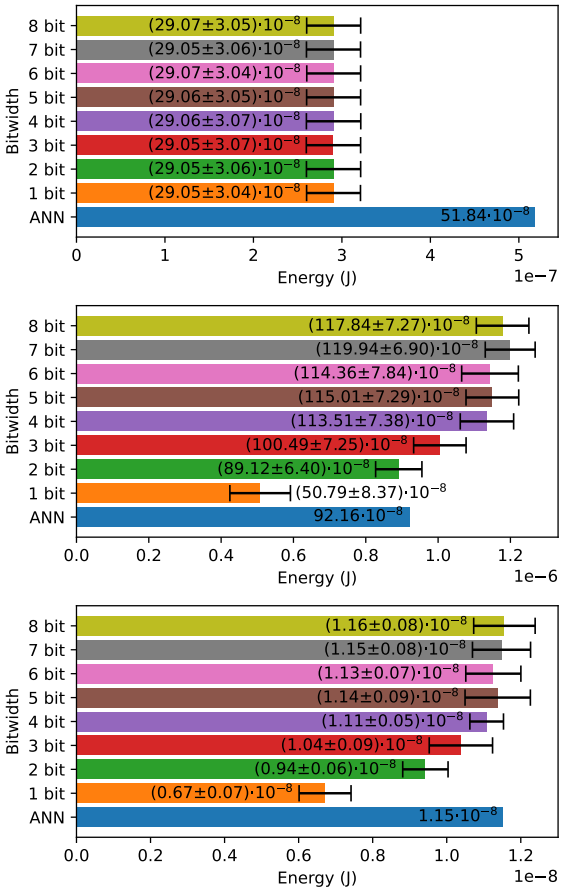
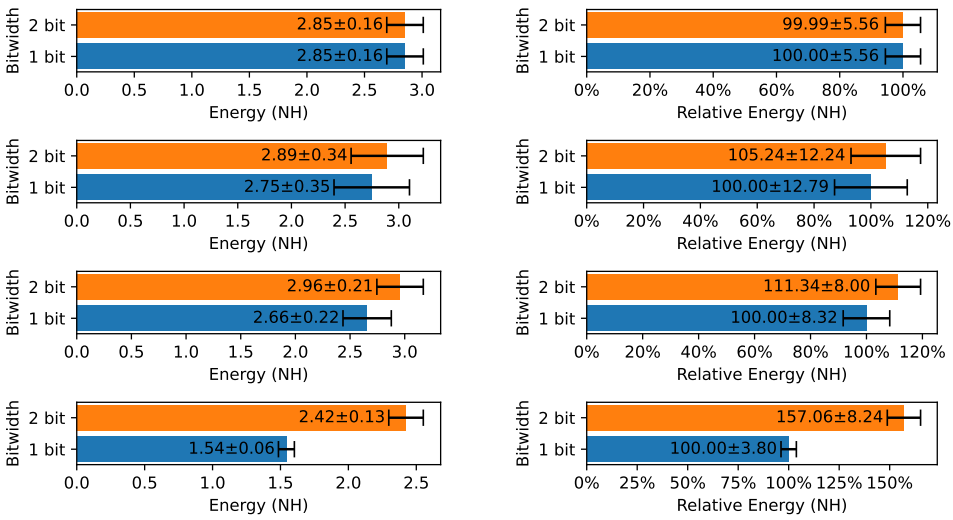


Figure 4.12: Comparison of the Energy Consumption of ANN and Multi-Bit Spike Train Model on Intel Loihi 2 for Fashion MNIST Dataset, conducted with 10 different random seeds, from top to bottom: first convolution block, second convolution block, output layer

4.6.3 Tradeoffs

One can tell that the energy consumption of the multi-bit spike train model has no direct advantage over the 1-bit spike train model on neuromorphic chips, as the firing rate of the multi-bit spike train model tends to be higher than the 1-bit spike train model.

We consider the energy consumption of the multi-bit spike train model in general as an opportunity to enable tradeoffs. If the inference is not the bottleneck of the application, then one can rely on the fast convergence speed of the multi-bit spike train model during training. If the inference is the bottleneck, then one can choose to train the multi-bit spike train model for longer time to achieve a firing rate that is comparable to the 1-bit spike train model (see Figure 4.13). Such tradeoffs are not possible with the 1-bit spike train model.



(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{AC}}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

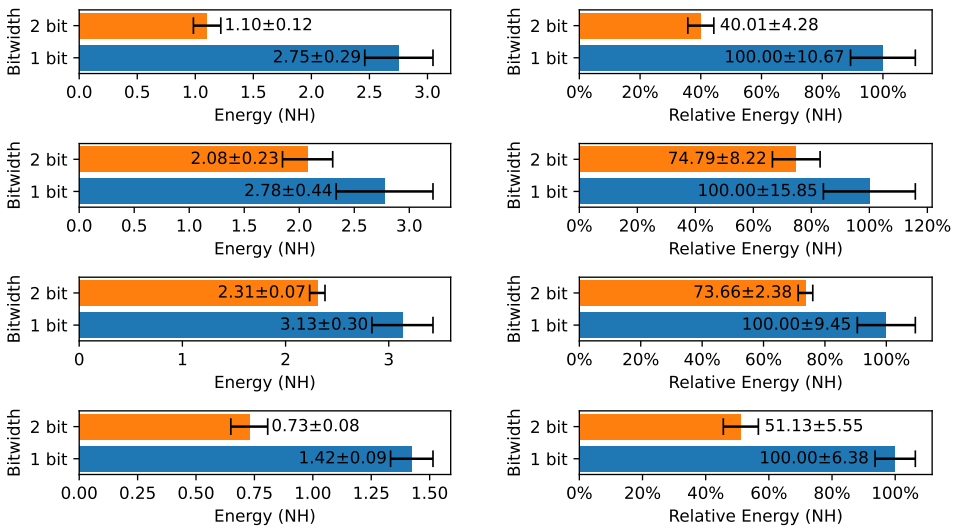
Figure 4.13: Inference Energy Consumption Estimation on Intel Loihi 2 for Fashion MNIST Dataset with 50 Training Epochs, conducted with 10 different random seeds

Additionally, if one is satisfied with the accuracy of the 1-bit spike train model, then one can choose to train the multi-bit spike train model for fewer time steps. This can enable higher efficiency in both training and inference.

We take again the example of the Fashion MNIST dataset, and while $T = 10$ is a good choice for both the 1-bit and 2-bit spike train model, we can reduce the time steps to $T = 4$ for the 2-bit spike train model and still achieve a comparable accuracy (see 4.14).

More details with additional results on the CIFAR10 dataset can be found in Appendix C.3.

The same tradeoffs allow the multi-bit spike train model to be more energy efficient than the ANN also for the Fashion MNIST dataset (see Figure 4.15).



(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{AC}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

(c) Accuracy Comparison

Figure 4.14: Inference Energy Consumption Estimation on Intel Loihi 2 and Test Accuracy for Fashion MNIST Dataset with 10 Time Steps for 1-bit Spike Train Model and 4 Time Steps for 2-bit Spike Train Model, conducted with 10 different random seeds

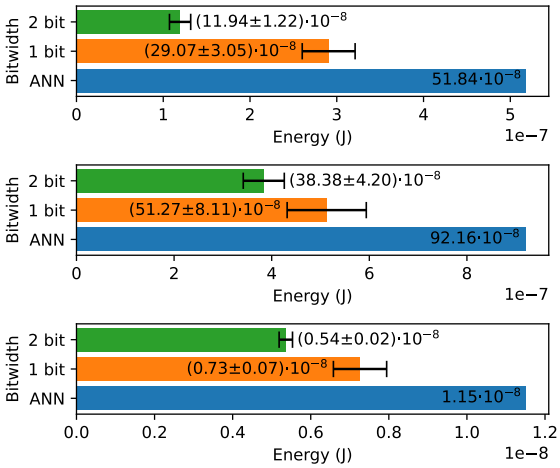


Figure 4.15: Comparison of the Energy Consumption of ANN and Multi-Bit Spike Train Model on Intel Loihi 2 for Fashion MNIST Dataset with 10 Time Steps for 1-bit Spike Train Model and 4 Time Steps for 2-bit Spike Train Model, from top to bottom: first convolution block, second convolution block, output layer

This can lead to a significant reduction in the energy consumption. It may not be reflected as a direct advantage in the energy consumption model mentioned above 4.6.2, but in practice, it should bring significant benefits, as most of the neuromorphic chips are designed to utilize the asynchronous communication via spikes, so they do not have a central clock system to synchronize the time steps very efficiently, and the cost for the synchronization barrier is very high. As reference, the latency per tile hop on Intel Loihi is at most around 6.5 ns where as the latency for the synchronization barrier is between 113 to 465 ns.

4.7 Performance

In the section 4.6.1, we claim that the energy consumption on the GPUs are not affected by the firing rate and the bit width of the spike train in theory. However, in practice, with the increase of the bit width of the spike train, the training process slows down. This could be caused by the inefficient implementation of the multi-bit spike train model and the low level optimization on operations of sparse matrix multiplication.

The performance limitation of the current implementation mainly comes from insufficient parallelization due to the time dependency in the temporal model and the lack of low level optimization on operations of low precision and sparse matrices. The temporal model enforces the time dependency in the network, precisely the time dependency of the membrane potential of the neurons. Thus, not only the forward path but also the backward path of the network must be executed sequentially. The frameworks like SpikingJelly and snnTorch, and their backend PyTorch, do not have sufficient support for sparse-dense matrix operations and mixed low precision operations (e.g. `fp8-bool` or `fp8-int2`).

The problems stated above can be considered as sufficient room for further optimization, e.g. utilizing the sparsity of the spike trains, supporting some low precision operations, and completely switching to the vectorized model instead of the temporal model instead, which is shown to be more efficient with learning algorithms like SLAYER and EXODUS.

Chapter 5

Related Work

We mainly used the LIF neuron model [6] in this thesis, which is a simple model that does not capture the full complexity of biological neurons. Hudgkin and Huxley [5] proposed a more detailed model that includes the dynamics of the ion channels, which is more biologically plausible but also computationally more expensive. Most of the models, e.g. Izhikevich's model [32], focus on some specific dynamics of the biological neurons and enable some tradeoffs.

Although BPTT is widely used in training deep large SNNs, other methods such as spike-timing-dependent plasticity (STDP) [33] are more biologically plausible with clear evidence in the brain. Methods like SLAYER [11] and EXODUS [12] focus more on the efficient computation of the gradients in the SNNs instead of the biological plausibility.

Typical constructions of SNNs include replacing the ReLU activation function in ANNs with the LIF neuron node. Due to the non-differentiability of the spike function and the time dependency of the membrane potential, the training of SNNs is more challenging than ANNs. Many methods, e.g. SpikeNorm [8] and Spiking RWKV [9] focus on replacing some components of ANNs with certain tweaks for SNNs to maintain the performance witnessed in ANNs while easing the training process.

As the demand for AI applications grows, the energy efficiency of the hardware becomes more important. Neuromorphic hardwares like IBM's TrueNorth [29] and Intel's Loihi [28] are designed to exploit the fact that within a discrete time step, there is no need for synchronous communication between the neurons, as each spike increases the membrane potential of the target neuron and such operation is commutative. This allows a partially asynchronous hardware design that can be more energy efficient than traditional GPUs. However, they lack the capability of training the network on the hardware, which is still done on traditional GPUs.

Quantization is a technique to reduce the memory footprint and computation cost of the neural networks by reducing the precision of the parameters. While large language models implemented in ANNs can be quantized to 4-bit precision [34], SNNs can be quantized to even lower precision [27] which should be able to achieve higher energy efficiency.

There exists work that shows the potential of using multi-bit spike trains in SNNs. In [35] the authors propose to use two integer number of bits to describe the ratio of the membrane potential to the threshold. They were able to show that the multi-bit spike train model can achieve better accuracy. Our work differs from [35] in that we divide the range of the membrane potential into $2^n - 1$ intervals and center the sigmoid function accordingly. This allows us to show the improvement in the convergence speed and the performance of the network.

Chapter 6

Concluding Remarks

6.1 Conclusion

In this thesis we have developed a novel spike train model for spiking neural networks (SNNs) that uses multi-bit spikes to encode the information. We implement it using an SNN framework, SpikingJelly, based on PyTorch. And we have shown that the multi-bit spike train model can significantly improve the convergence speed by around 50% with the 2-bit setup and slightly better accuracy of the network compared to the traditional 1-bit spike train model while preserving other characteristics of the 1-bit spike train model such as high quantizability and low energy consumption. We consider the tradeoffs of the multi-bit spike train model can bring in terms of energy consumption and performance important for maximizing the efficiency of SNNs on various hardware platforms. Specifically on Fashion MNIST and CIFAR10, we are able to achieve up to 60% energy consumption reduction on neuromorphic hardware with the 2-bit spike train model compared to the 1-bit spike train model.

All the code and experiments are available at `github.com/SkyWorld117/MultibitSpikes`.

6.2 Future Work

The multi-bit spike train model is a promising direction for the development of SNNs. However, there are still many open questions and challenges that need to be addressed in the future. Here we list some of the possible future work:

- **Optimization of the multi-bit spike train model:** The current implementation of the multi-bit spike train model is not optimized for

performance. One can consider using just-in-time (JIT) compilation to improve the performance of the model.

- **Investigation of the overfitting problem:** The multi-bit spike train model is more complicated than the 1-bit spike train model, which can lead to overfitting. One can investigate the overfitting problem and propose solutions to mitigate it.
- **Extension to other tasks and datasets:** The experiments in this thesis are mainly focused on image classification tasks. One can extend the multi-bit spike train model to other tasks and datasets to evaluate its performance.
- **Implementation on neuromorphic chips:** The multi-bit spike train model is designed to be hardware-friendly in theory. It would be more convincing if one can implement the model on neuromorphic chips like Intel Loihi 2 to evaluate its performance on specialized hardware.
- **Investigation of the energy consumption model:** The energy consumption model presented in this thesis is a simple estimation. One can investigate the energy consumption of the multi-bit spike train model more thoroughly and propose a more accurate model. Ideally, one can also measure the energy consumption of the multi-bit spike train model after implementing it on neuromorphic chips.

Appendix A

Accuracy of the Multi-Bit Spike Train Model

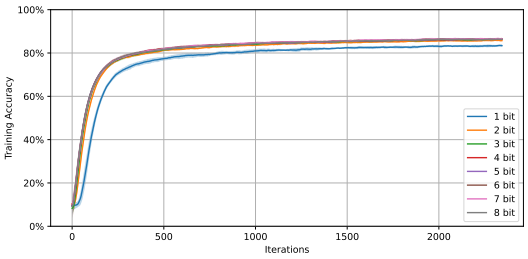
A.1 Accuracy Curves

All train accuracy curves are smoothed with a window size of 100.

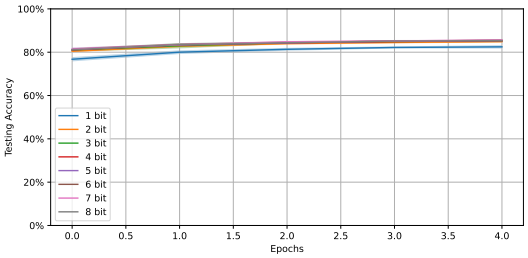
The hyperparameters used in the training process are shown in Table A.1.

Dataset	Reps	Epochs	LR	Opt	Batch	Time steps
Fashion MNIST	10	5	2e-3	Adam	128	10
MNIST	10	5	2e-3	Adam	128	10
NMNIST	10	5	2e-3	Adam	128	10
DVS Gesture	10	20	1e-3	Adam	128	10
CIFAR-10	5	50	1e-5	Adam	128	10

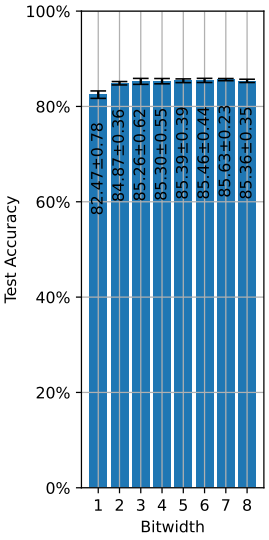
Table A.1: Hyperparameters



(a) Train Accuracy

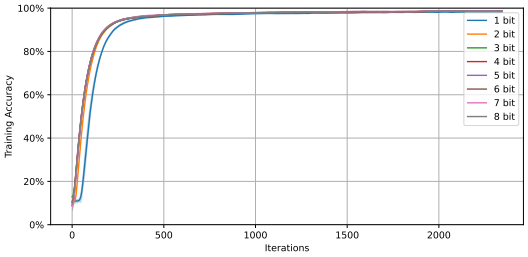


(b) Test Accuracy

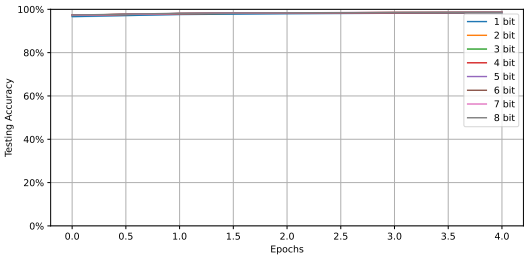


(c) Final Test Accuracy

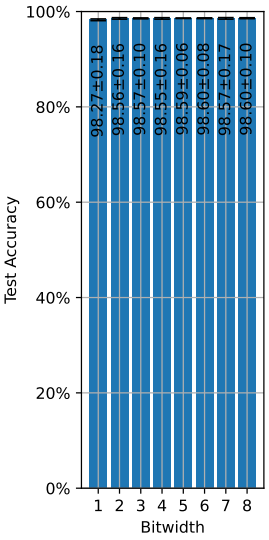
Figure A.1: Accuracy Curves of the Fashion MNIST Model



(a) Train Accuracy

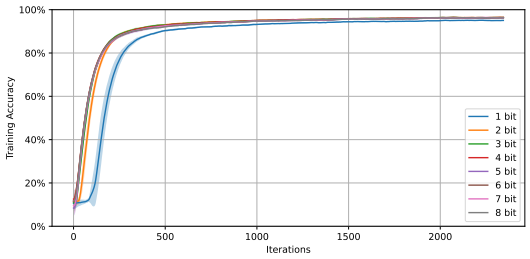


(b) Test Accuracy

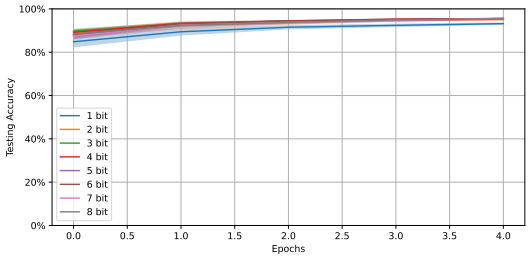


(c) Final Test Accuracy

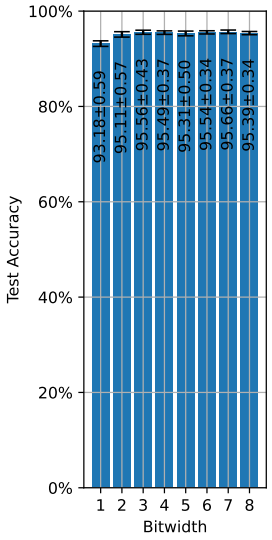
Figure A.2: Accuracy Curves of the MNIST Model



(a) Train Accuracy

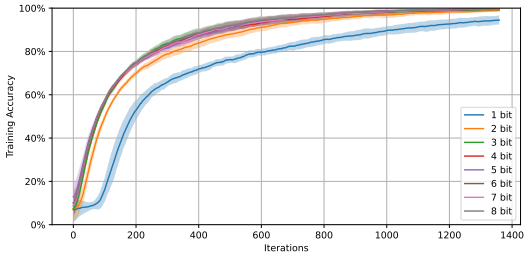


(b) Test Accuracy

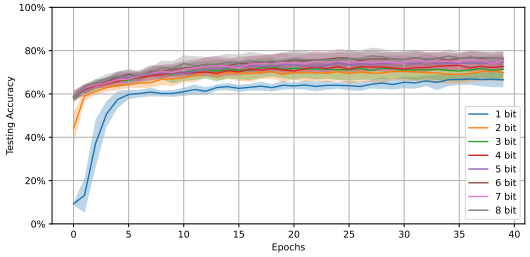


(c) Final Test Accuracy

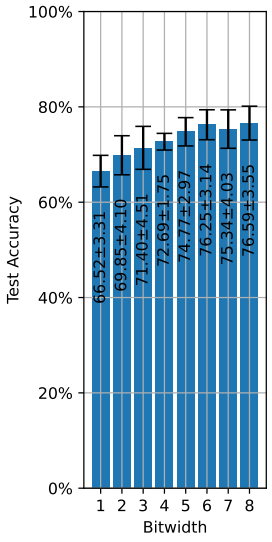
Figure A.3: Accuracy Curves of the NMNIST Model



(a) Train Accuracy

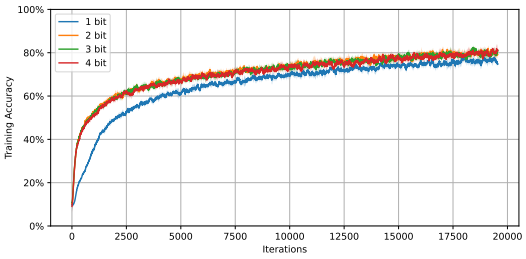


(b) Test Accuracy

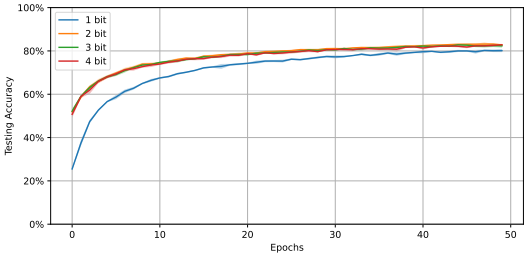


(c) Final Test Accuracy

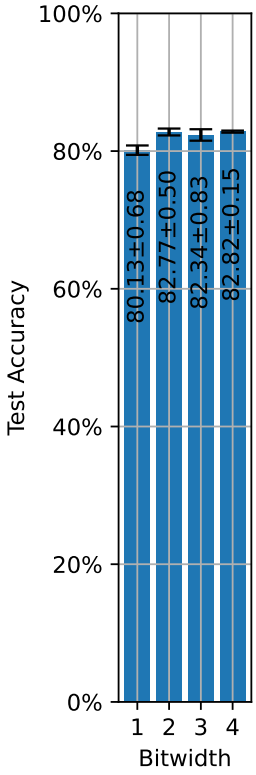
Figure A.4: Accuracy Curves of the DVS Gesture Model



(a) Train Accuracy



(b) Test Accuracy



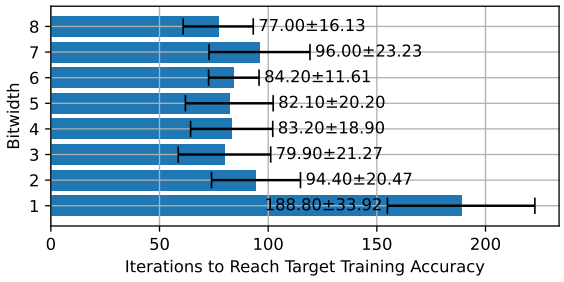
(c) Final Test Accuracy

Figure A.5: Accuracy Curves of the CIFAR-10 Model

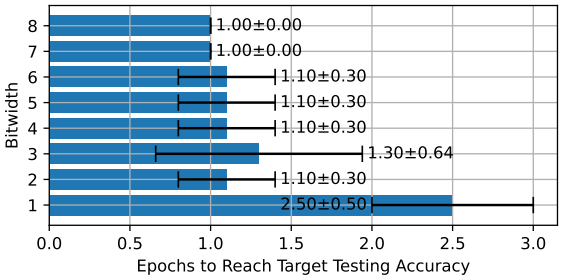
A.2 Iterations and Epochs to Reach the Target Accuracy

In this section, we consider the target accuracy to be 80%.

Hyperparameters: Same as in Table A.1.

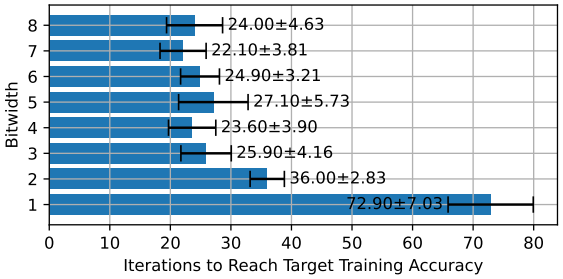


(a) Iterations to Reach the Training Accuracy

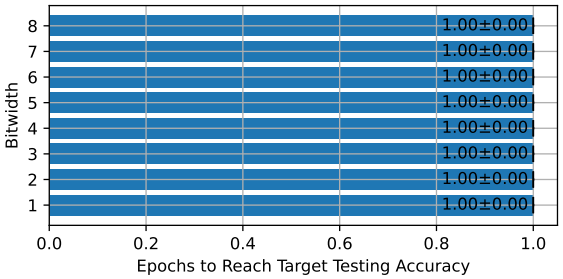


(b) Epochs to Reach the Test Accuracy

Figure A.6: Required Iterations of Epochs on Fashion MNIST

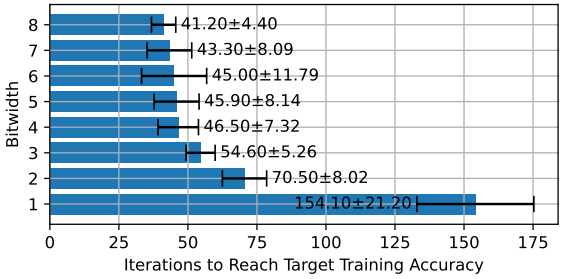


(a) Iterations to Reach the Training Accuracy

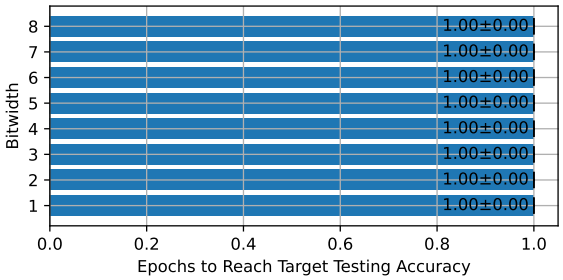


(b) Epochs to Reach the Test Accuracy

Figure A.7: Required Iterations of Epochs on MNIST

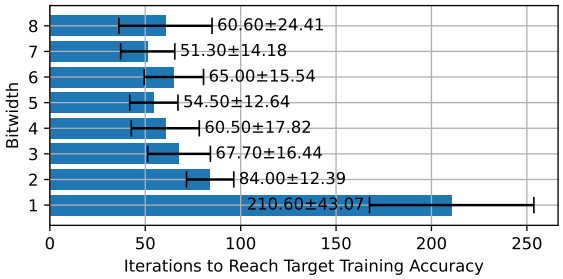


(a) Iterations to Reach the Training Accuracy

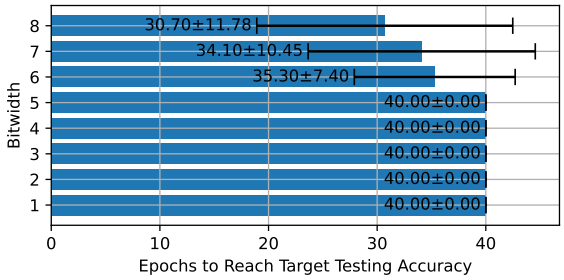


(b) Epochs to Reach the Test Accuracy

Figure A.8: Required Iterations of Epochs on NMNIST



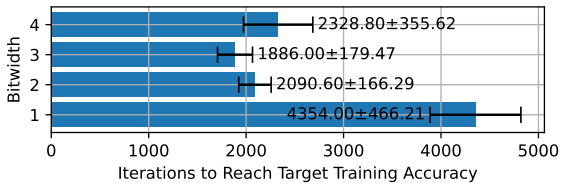
(a) Iterations to Reach the Training Accuracy



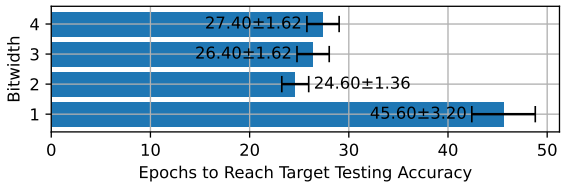
(b) Epochs to Reach the Test Accuracy

Figure A.9: Required Iterations of Epochs on DVS Gesture

Notice that we encountered overfitting in the DVS Gesture dataset, which is why the number of iterations to reach the target test accuracy is capped at 40 epochs (the maximum number of epochs in the training process), thus not representative.



(a) Iterations to Reach the Training Accuracy



(b) Epochs to Reach the Test Accuracy

Figure A.10: Required Iterations of Epochs on CIFAR-10

Appendix B

Firing Rate in Different Positions of the Multi-Bit Spike Train Model

For the ease of debugging, the first measuring position is always just the input data. If the data is captured by a dynamic vision sensor or it is encoded with a Poisson spike generator, one should see a firing rate much below 100%. In the case of original data (e.g. CIFAR-10), the firing rate should be close to 100%.

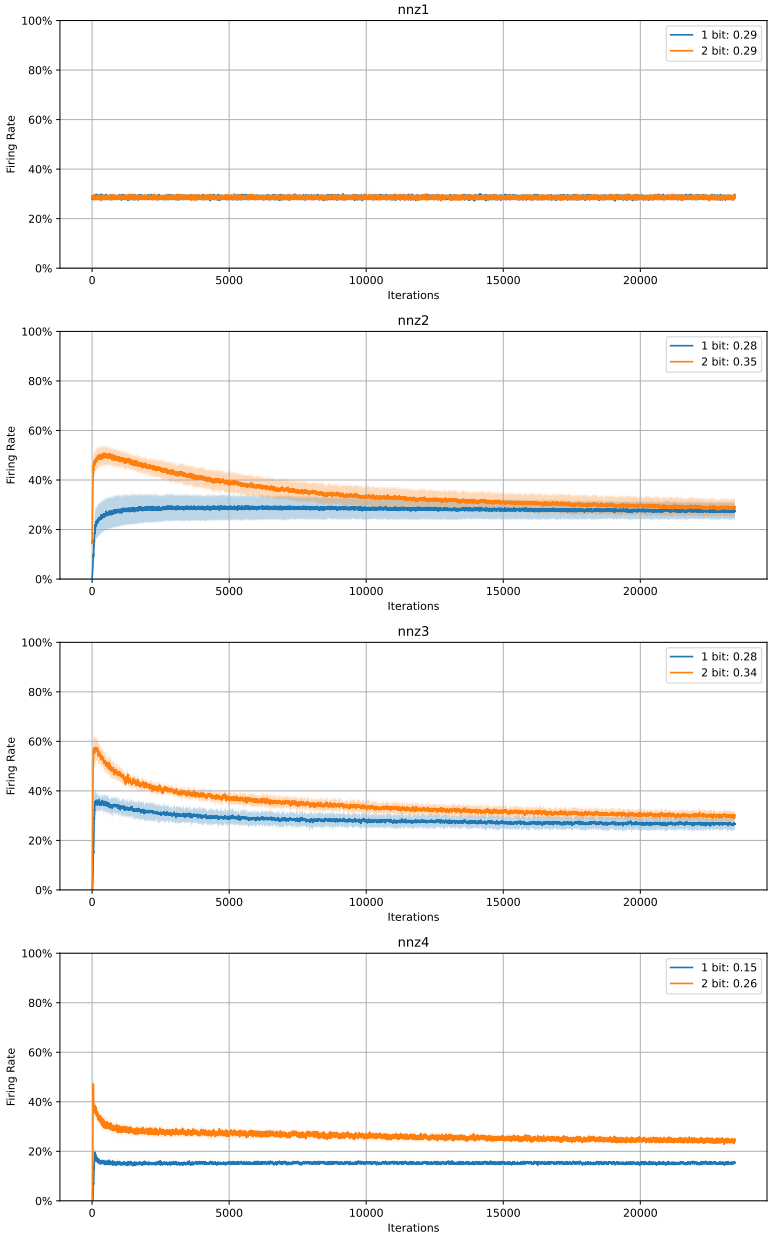
From the second measuring position to the last position, the firing rate is measured after the LIF neuron layers. Not all LIF neuron layers are measured. But the last LIF neuron layer is always measured.

The hyperparameters used in the training process are shown in Table B.1.

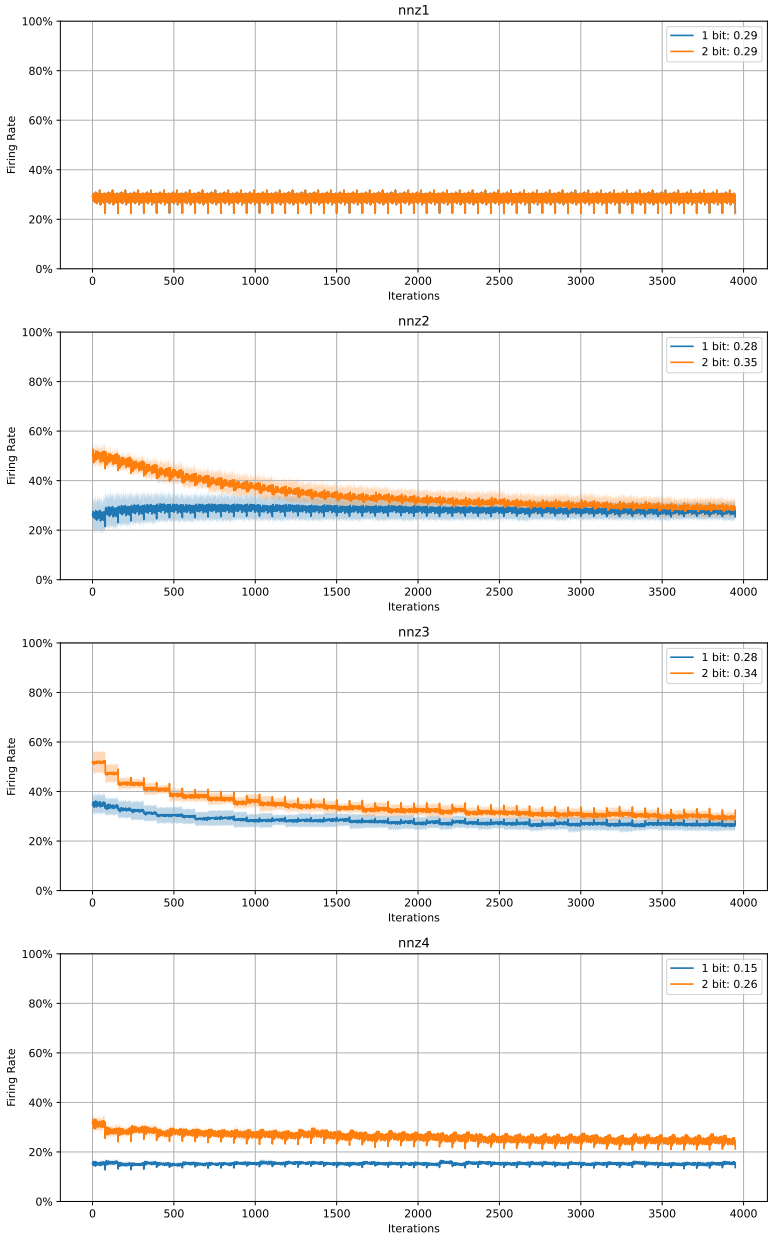
Dataset	Reps	Epochs	LR	Opt	Batch	Time steps
Fashion MNIST	10	50	2e-3	Adam	128	10
MNIST	10	50	2e-3	Adam	128	10
NMNIST	10	50	2e-3	Adam	128	10
DVS Gesture	10	200	1e-3	Adam	128	10
CIFAR-10	5	1000	1e-5	Adam	128	10

Table B.1: Hyperparameters

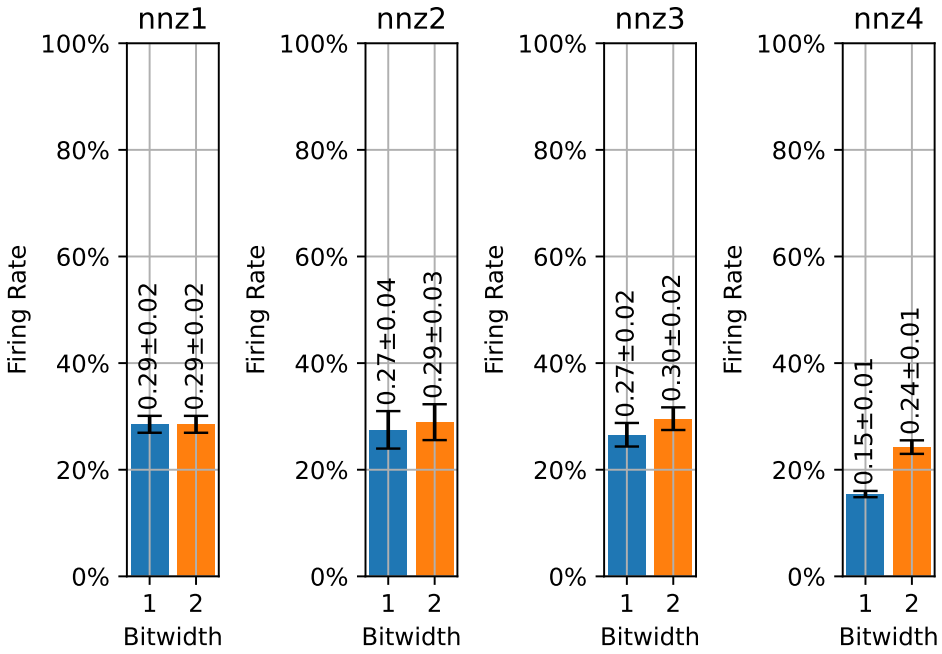
B.1 Fashion MNIST



(a) Firing Rate in Different Positions (Training)



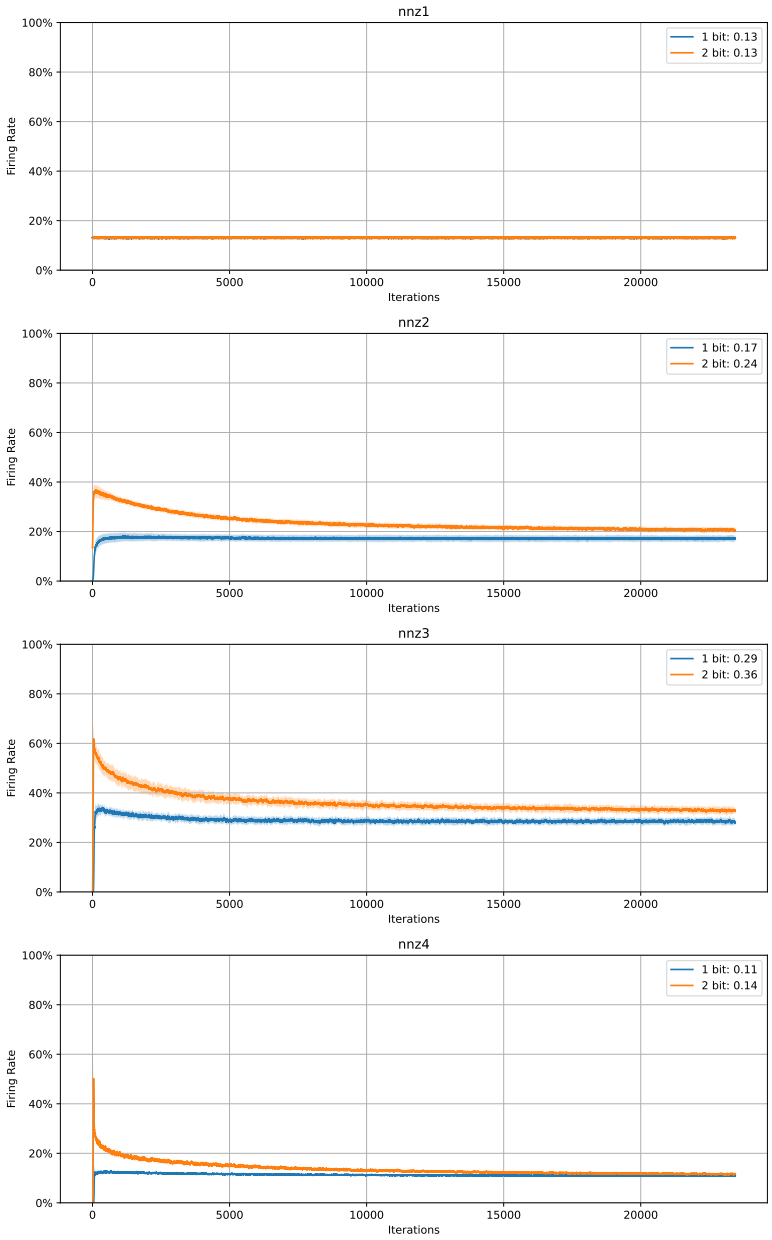
(b) Firing Rate in Different Positions (Test)



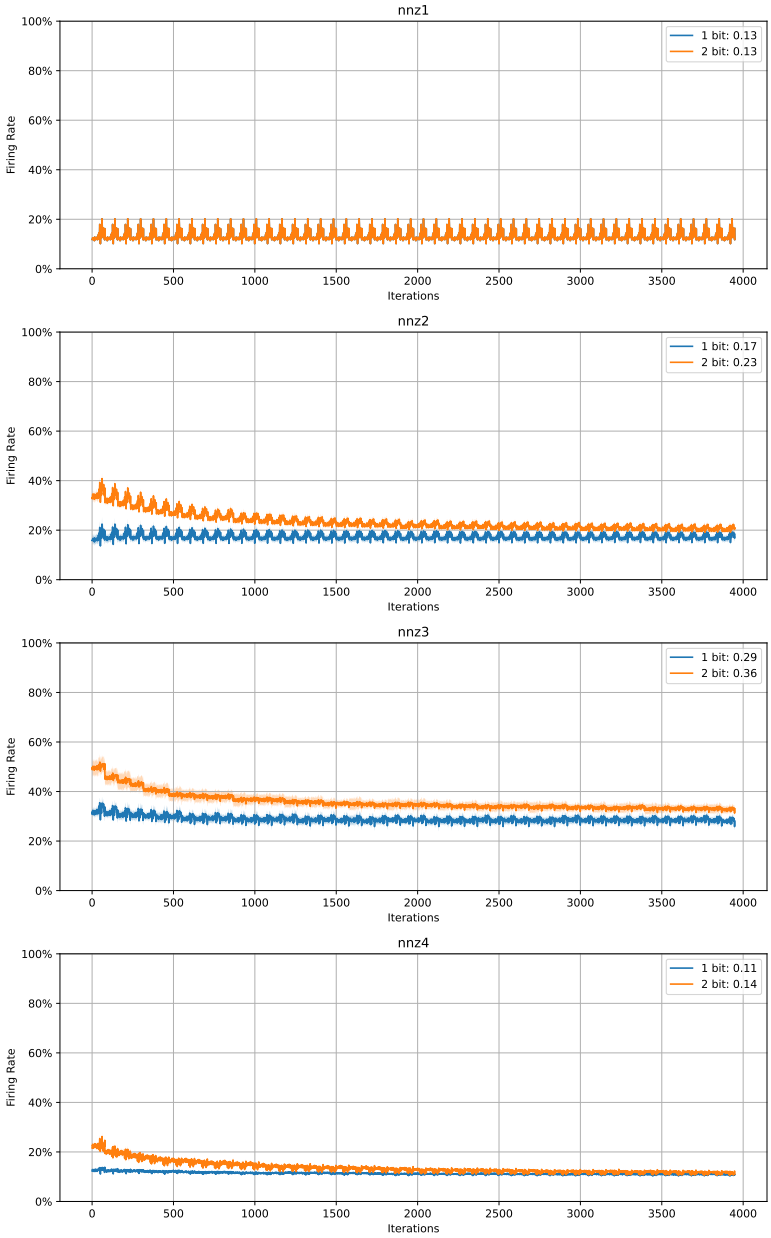
(c) Firing Rate in Different Positions (Final Test)

Figure B.1: Firing Rate in Different Positions of the Fashion MNIST Model

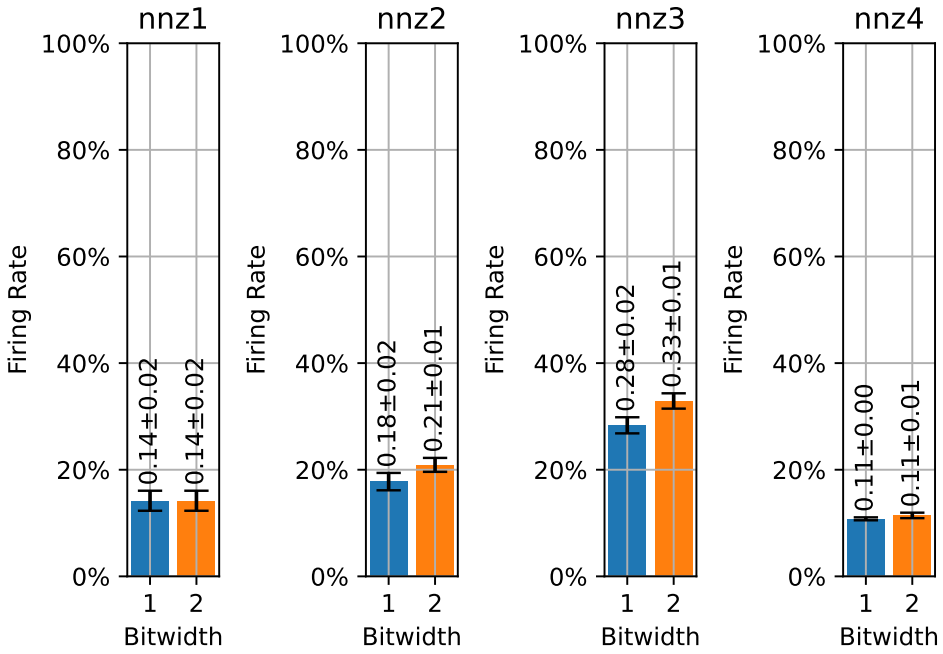
B.2 MNIST



(a) Firing Rate in Different Positions (Training)



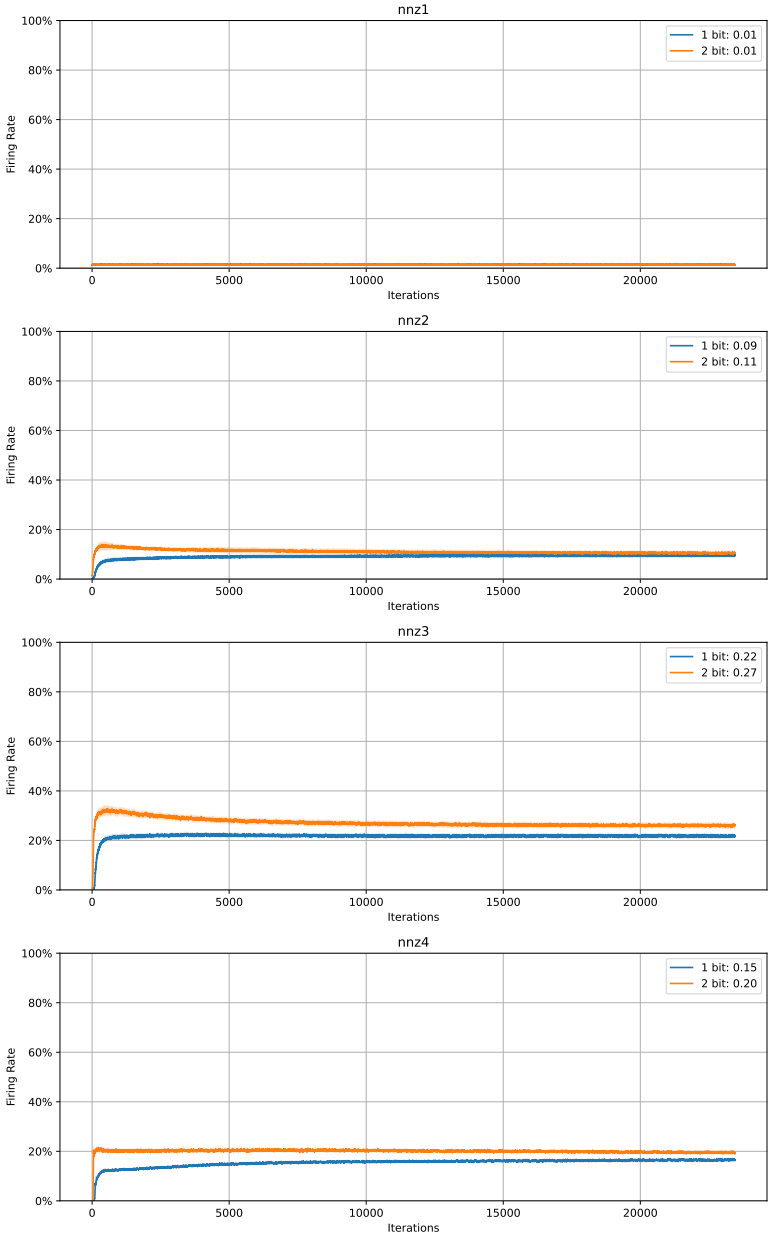
(b) Firing Rate in Different Positions (Test)



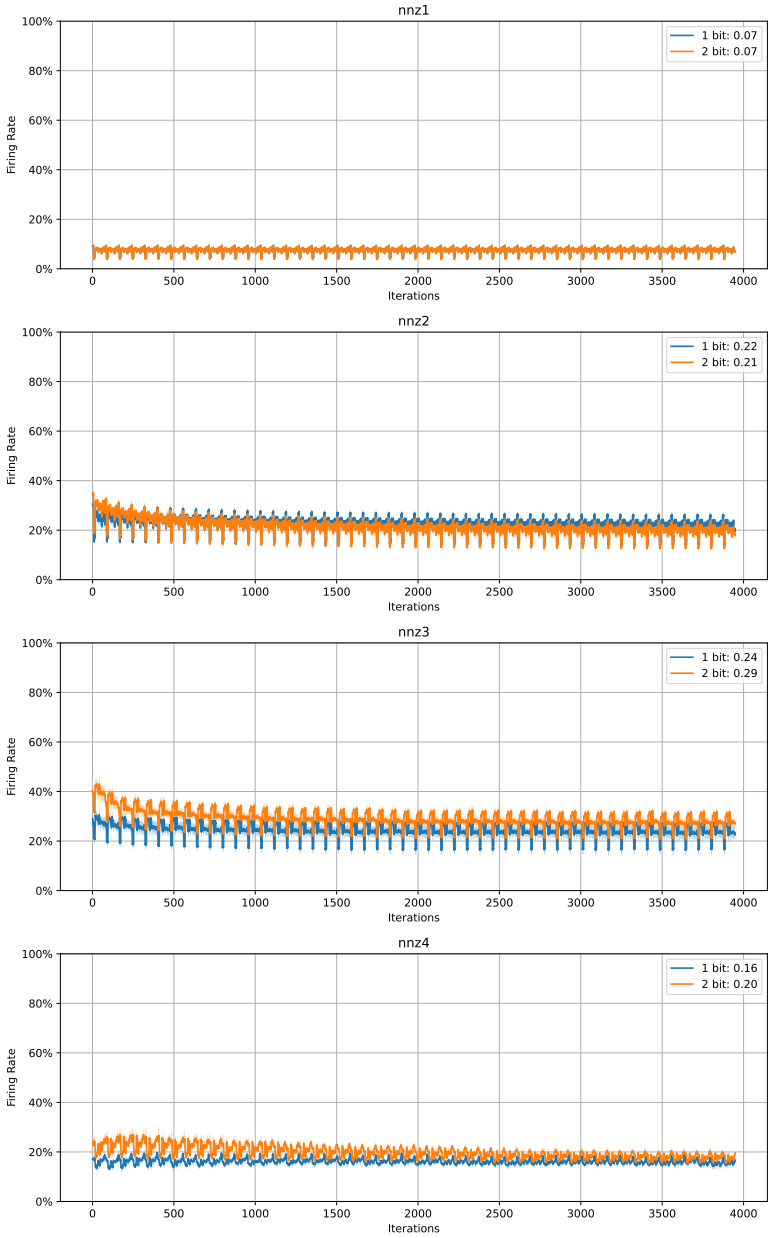
(c) Firing Rate in Different Positions (Final Test)

Figure B.2: Firing Rate in Different Positions of the MNIST Model

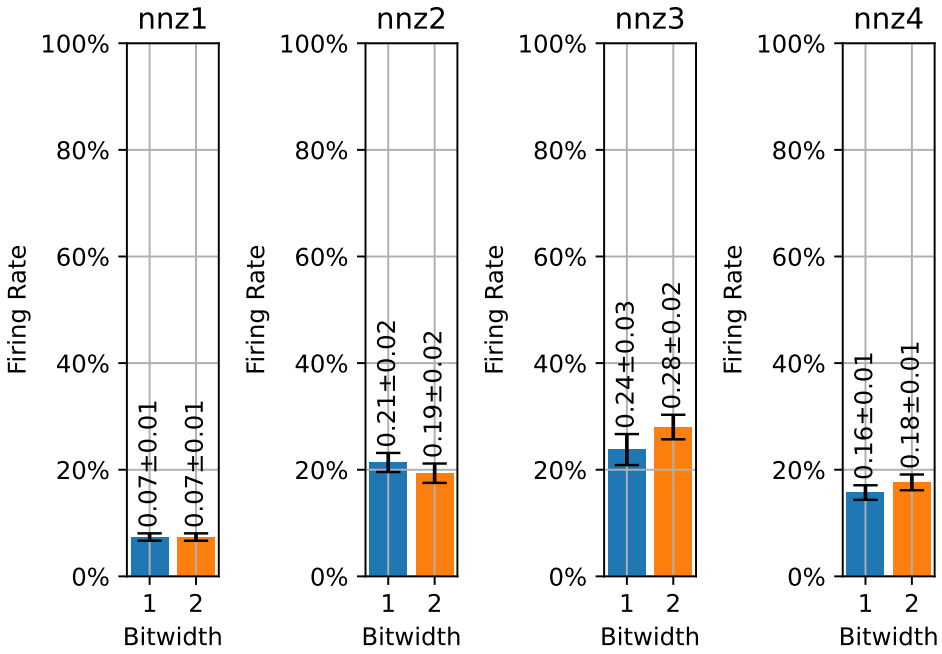
B.3 NMNIST



(a) Firing Rate in Different Positions (Training)



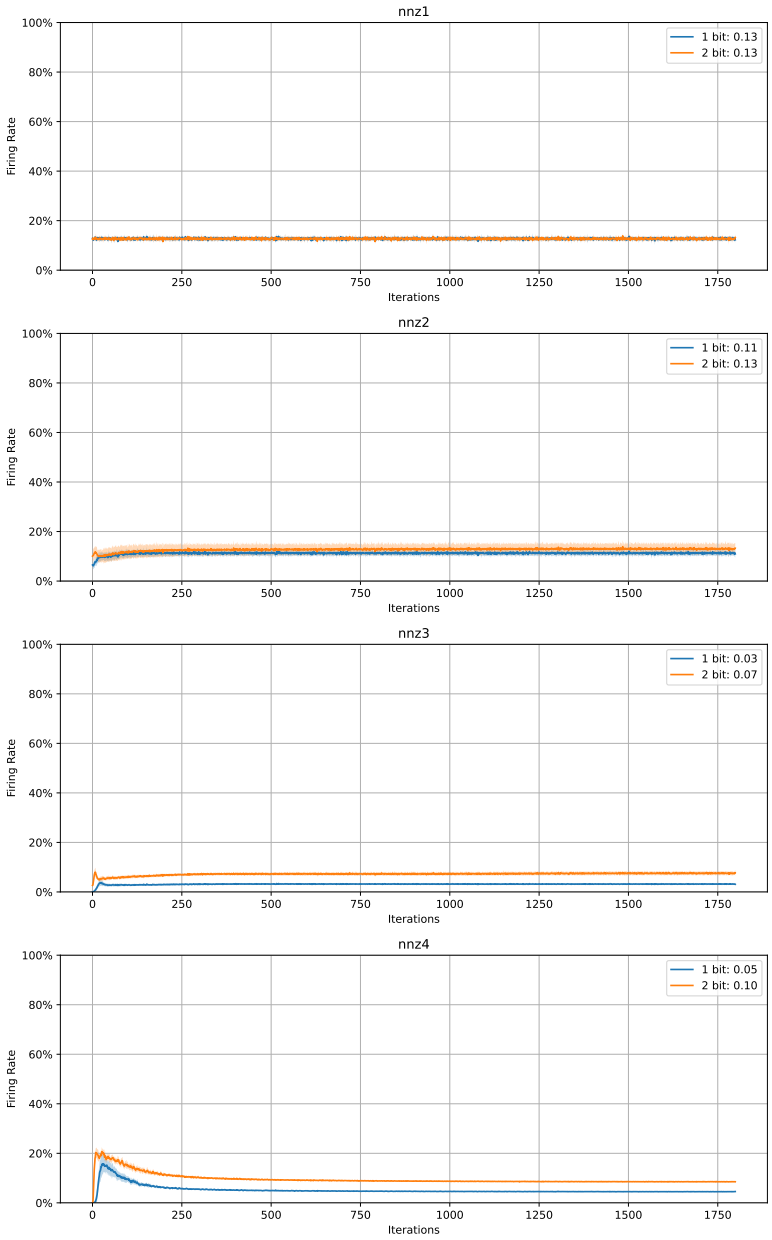
(b) Firing Rate in Different Positions (Test)



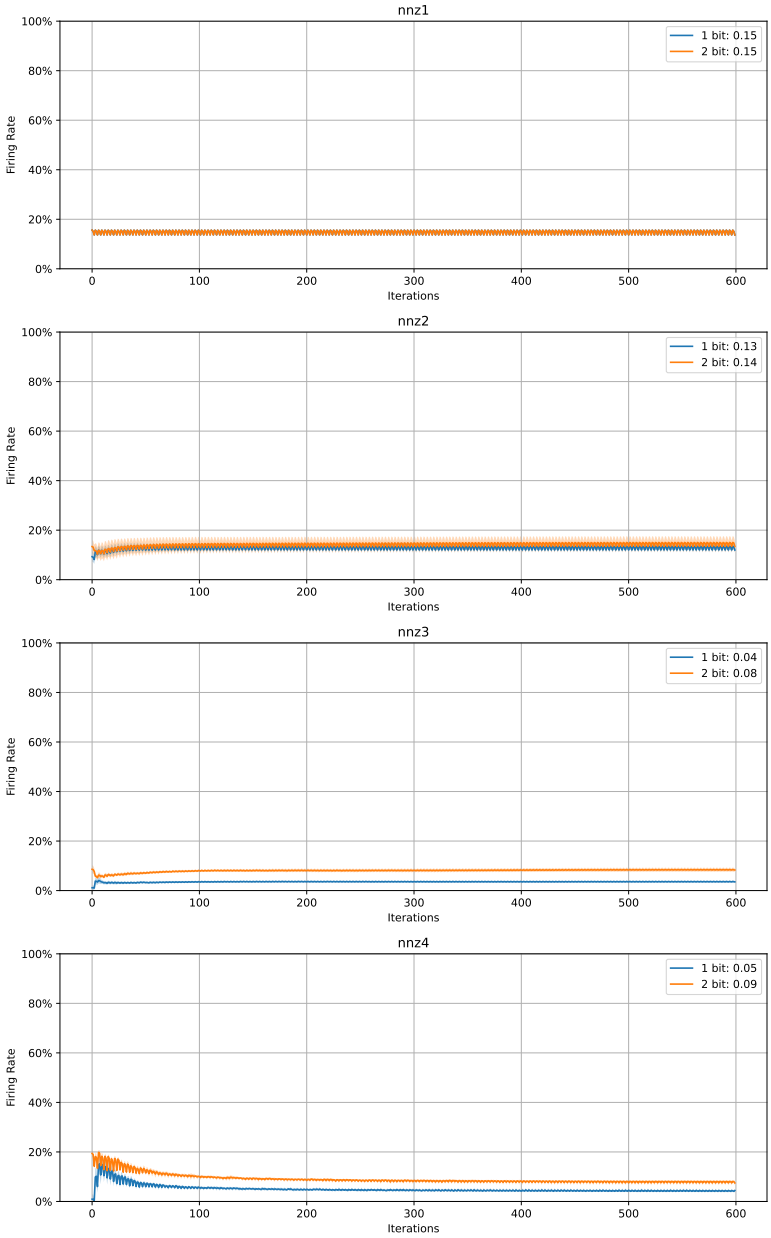
(c) Firing Rate in Different Positions (Final Test)

Figure B.3: Firing Rate in Different Positions of the NMNIST Model

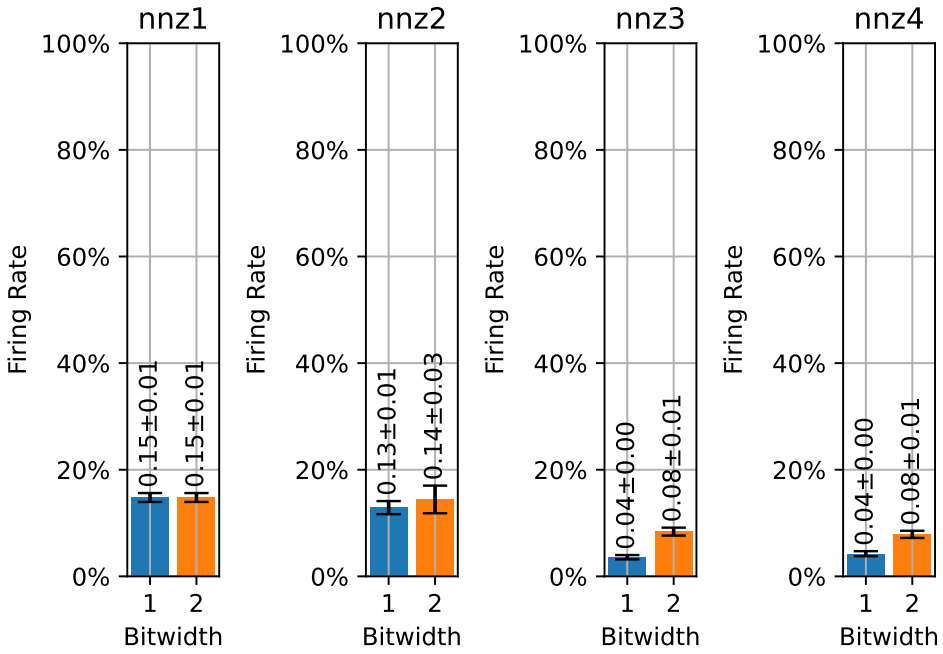
B.4 DVS Gesture



(a) Firing Rate in Different Positions (Training)



(b) Firing Rate in Different Positions (Test)

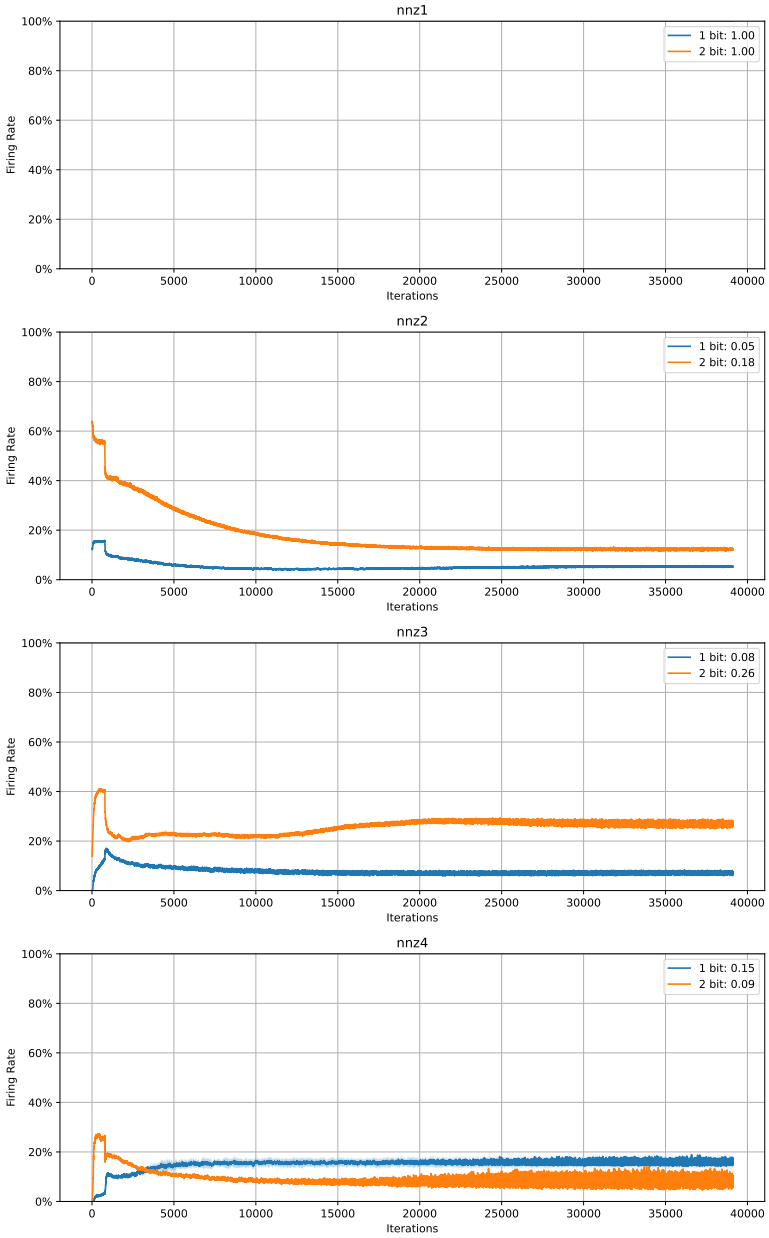


(c) Firing Rate in Different Positions (Final Test)

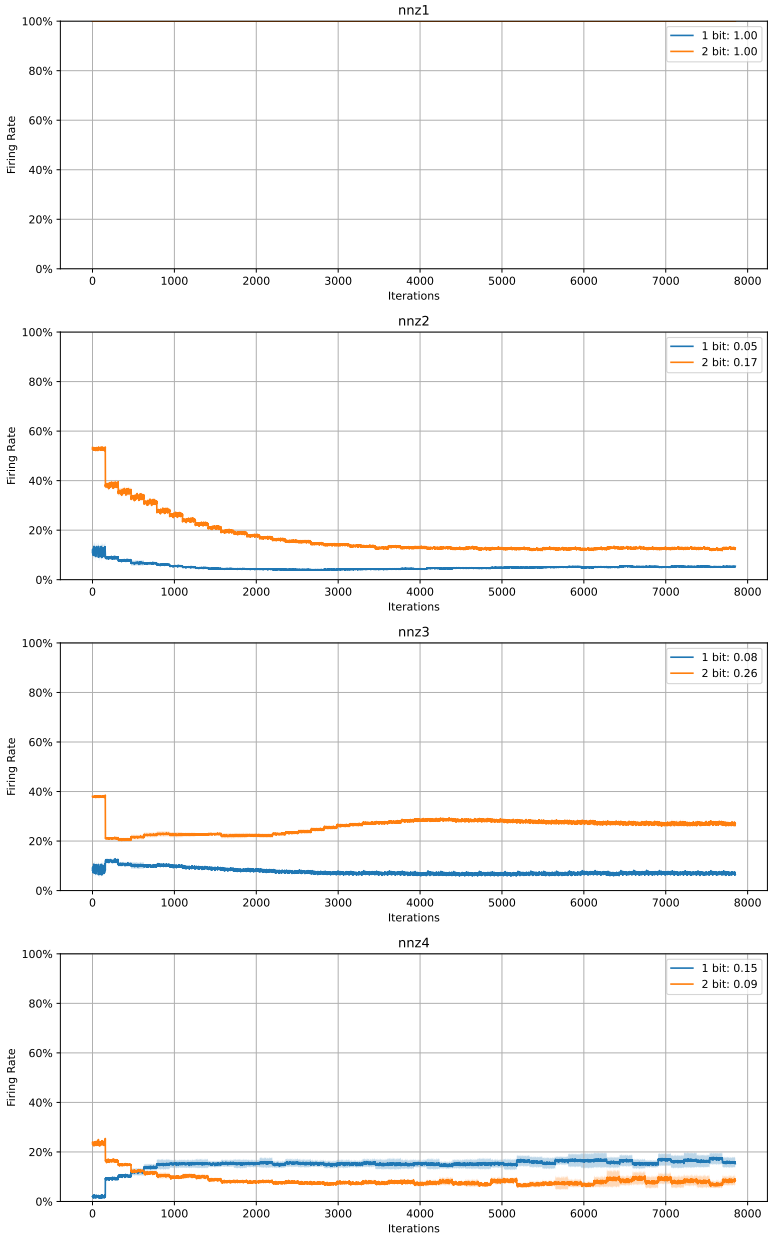
Figure B.4: Firing Rate in Different Positions of the DVS Gesture Model

B.5 CIFAR-10

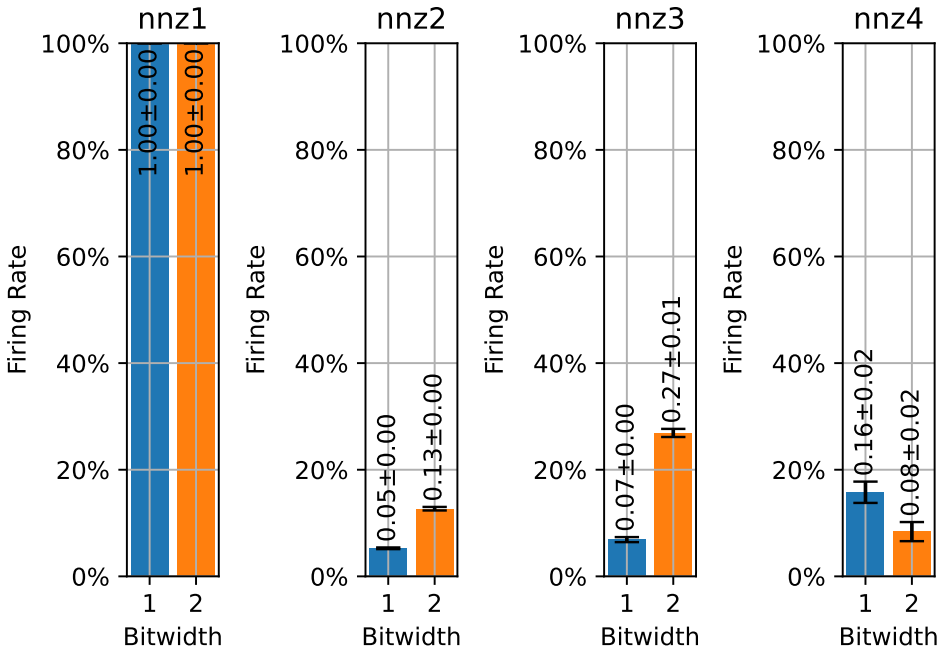
Here we use learning rate schedulers to accelerate the convergence of the models. Although the models can reach a high accuracy faster, the multi-bit spike train model suffers from the high learning rate from time to time.



(a) Firing Rate in Different Positions (Training)



(b) Firing Rate in Different Positions (Test)



(c) Firing Rate in Different Positions (Final Test)

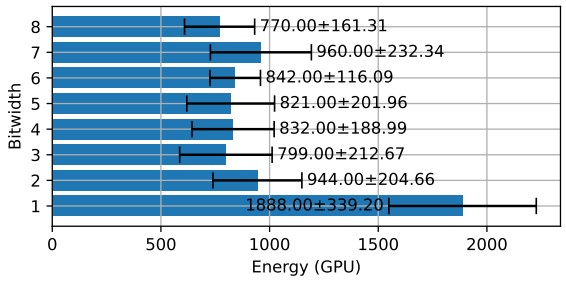
Figure B.5: Firing Rate in Different Positions of the CIFAR-10 Model

Appendix C

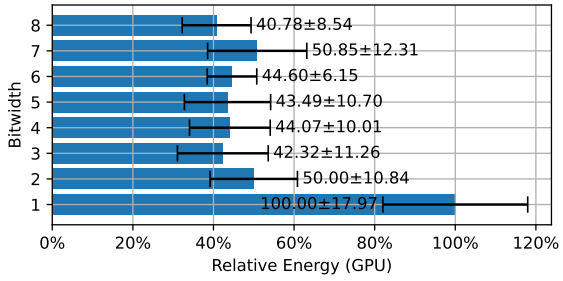
Energy Consumption Estimation

C.1 Training Energy Consumption Estimation on GPUs

The hyperparameters used in the training process are the same as in Table A.1.

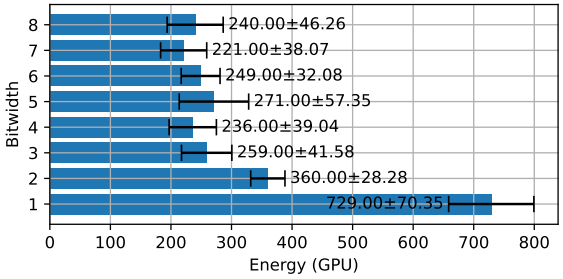


(a) Energy Consumption Estimation, Unit in Constant c

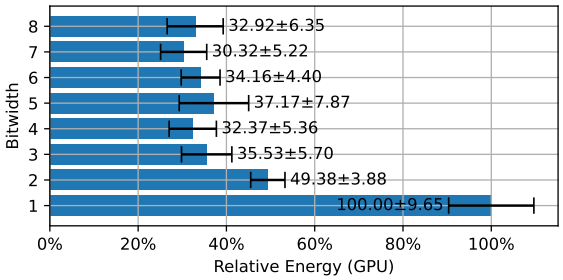


(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.1: Training Energy Consumption Estimation on GPUs for Fashion MNIST Dataset

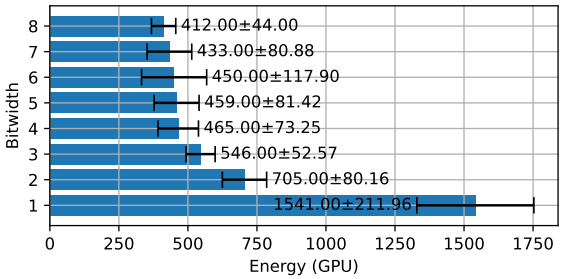


(a) Energy Consumption Estimation, Unit in Constant c

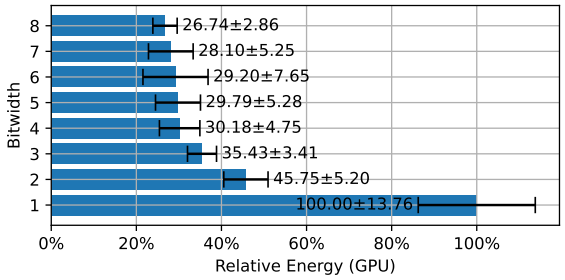


(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.2: Training Energy Consumption Estimation on GPUs for MNIST Dataset

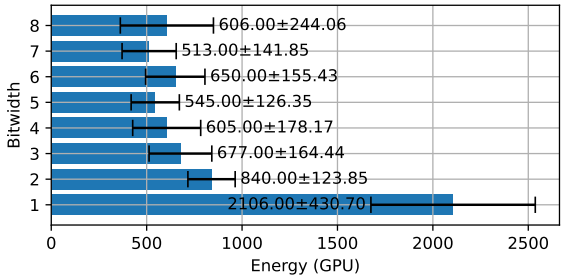


(a) Energy Consumption Estimation, Unit in Constant c

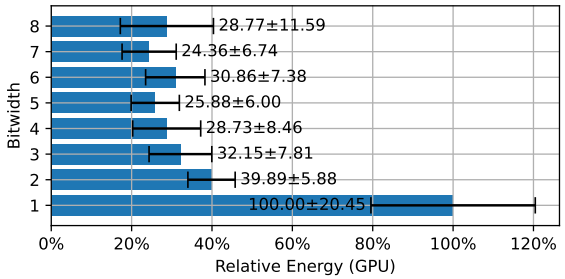


(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.3: Training Energy Consumption Estimation on GPUs for NMNIST Dataset

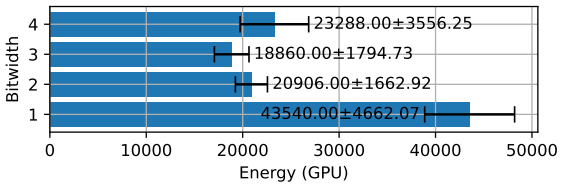


(a) Energy Consumption Estimation, Unit in Constant c

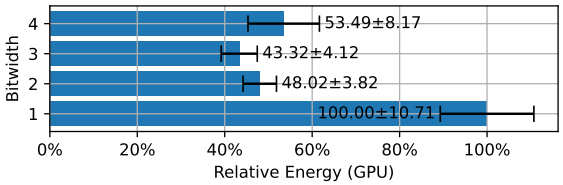


(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.4: Training Energy Consumption Estimation on GPUs for DVS Gesture Dataset



(a) Energy Consumption Estimation, Unit in Constant c



(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

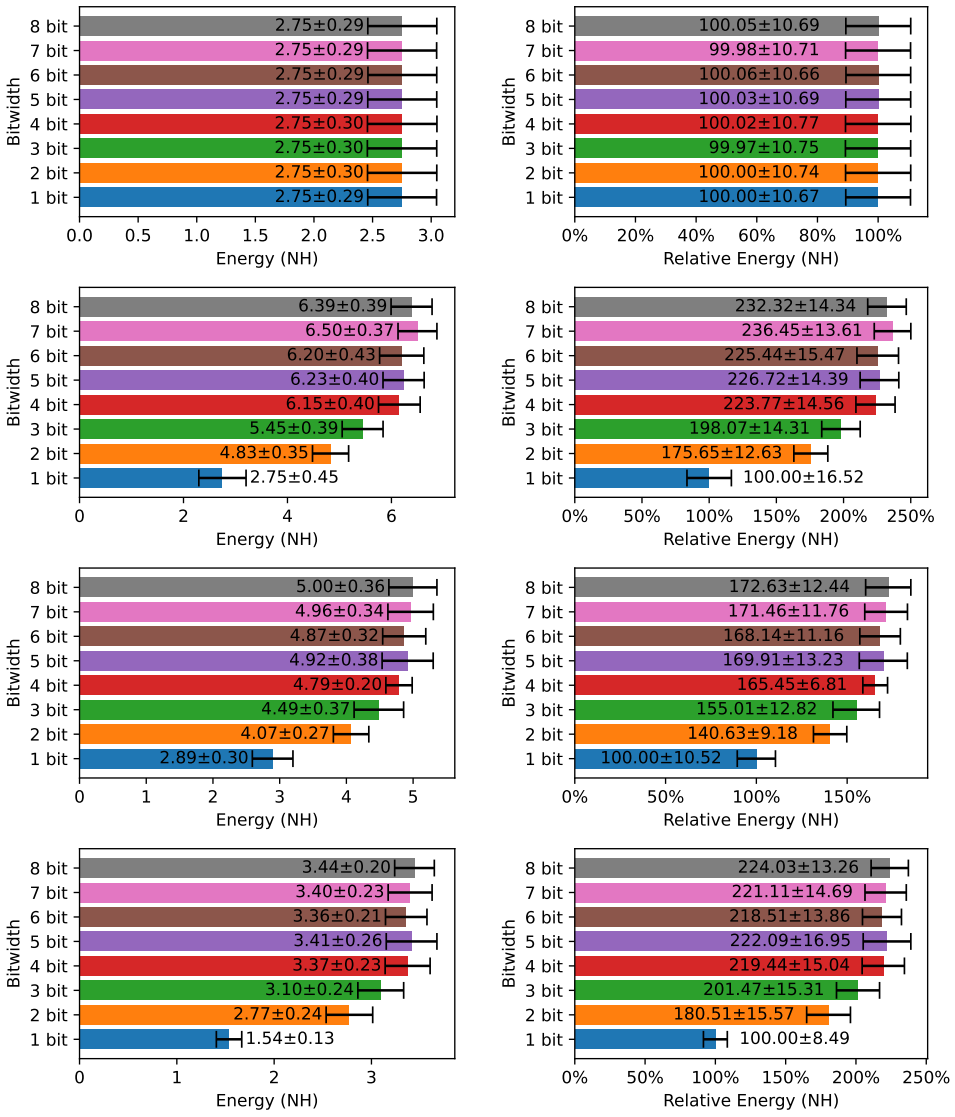
Figure C.5: Training Energy Consumption Estimation on GPUs for CIFAR-10 Dataset

C.2 Inference Energy Consumption Estimation on Neuromorphic Hardware

We assume the models are running on neuromorphic hardware like Intel Loihi 2 that support graded spikes.

The hyperparameters used in the training process are the same as in Table A.1.

C.2.1 Fashion MNIST

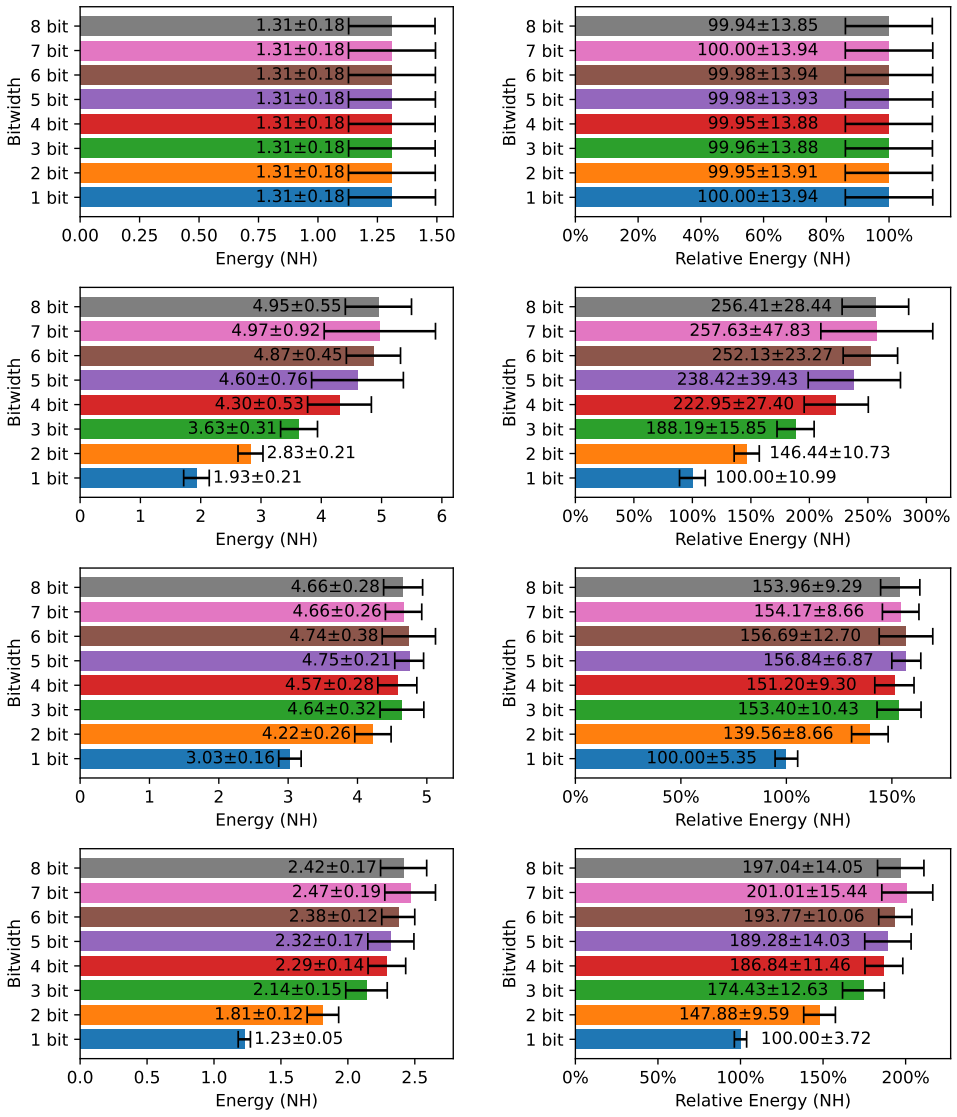


(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.6: Inference Energy Consumption Estimation on Intel Loihi 2

C.2.2 MNIST

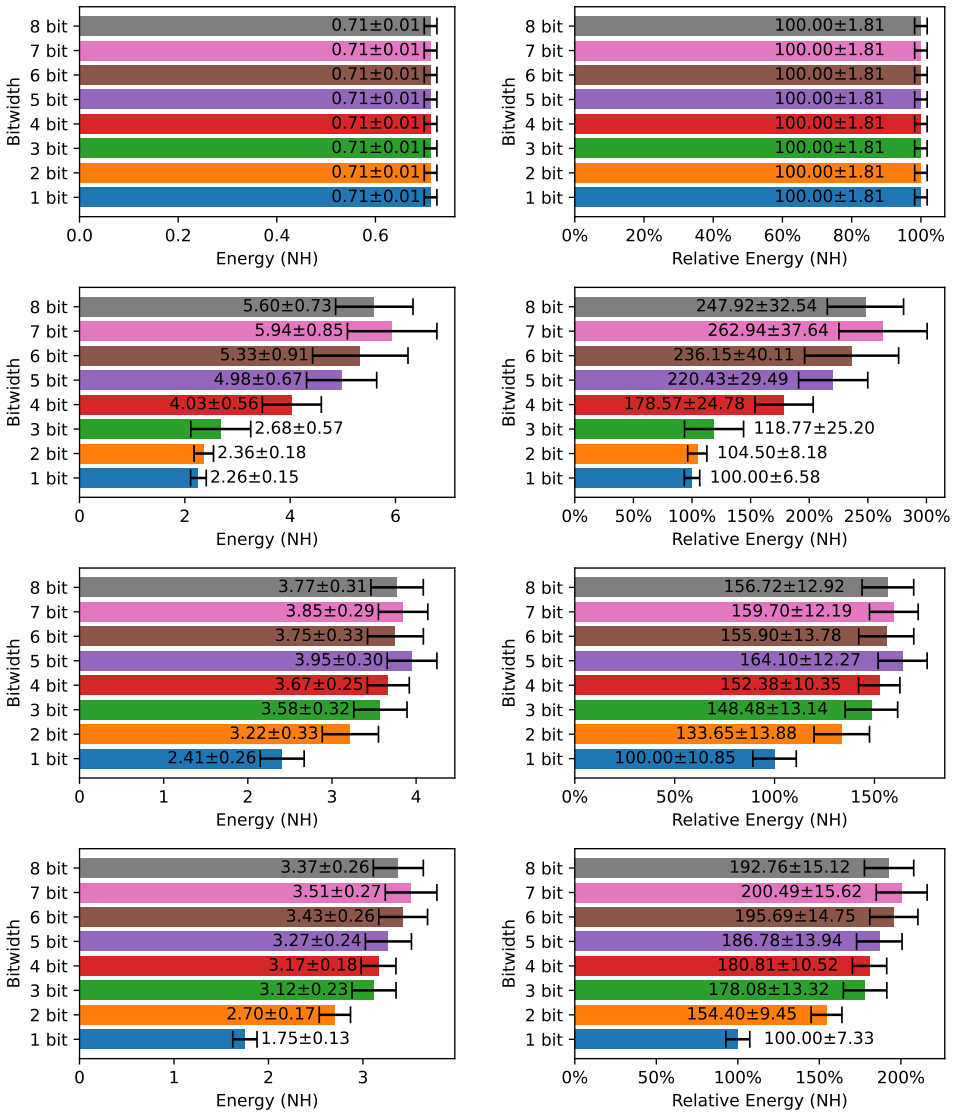


(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.7: Inference Energy Consumption Estimation on Intel Loihi 2

C.2.3 NMNIST

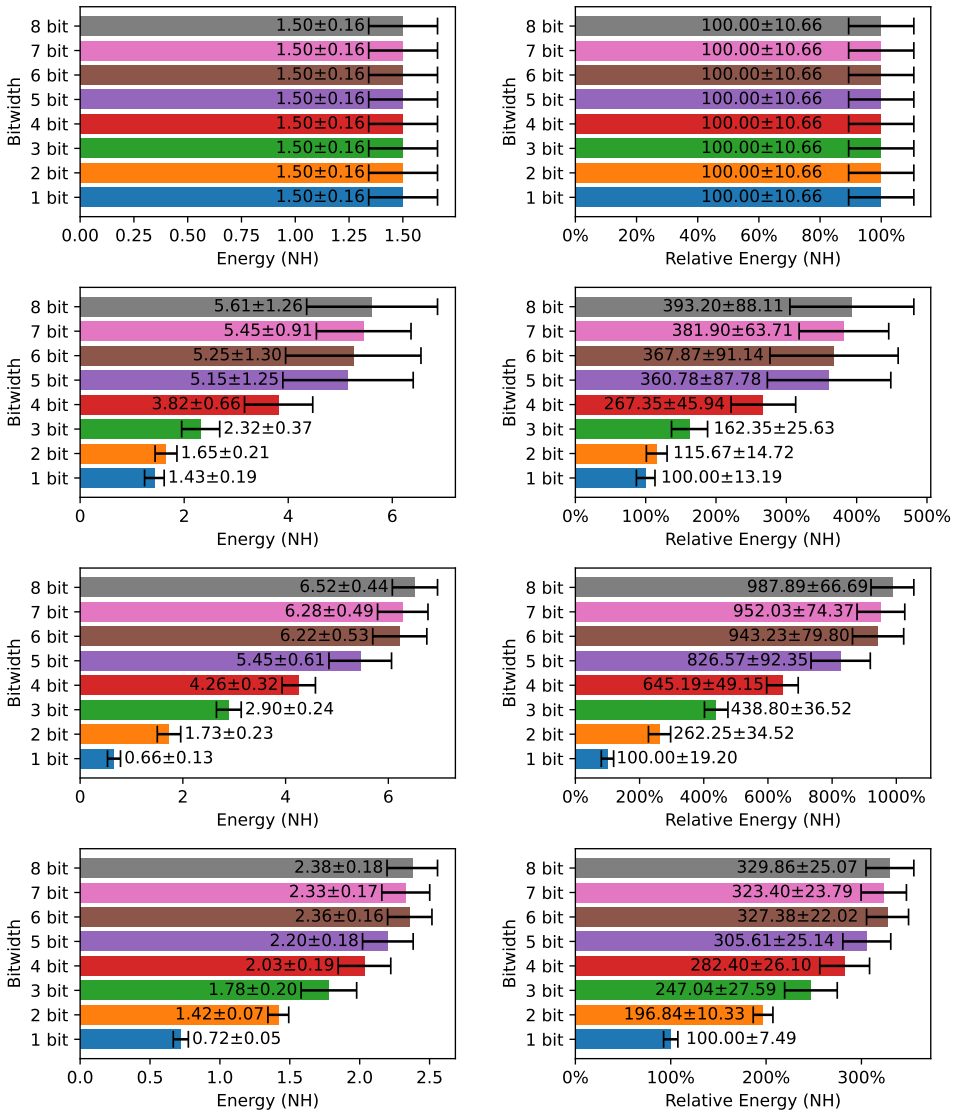


(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.8: Inference Energy Consumption Estimation on Intel Loihi 2

C.2.4 DVS Gesture

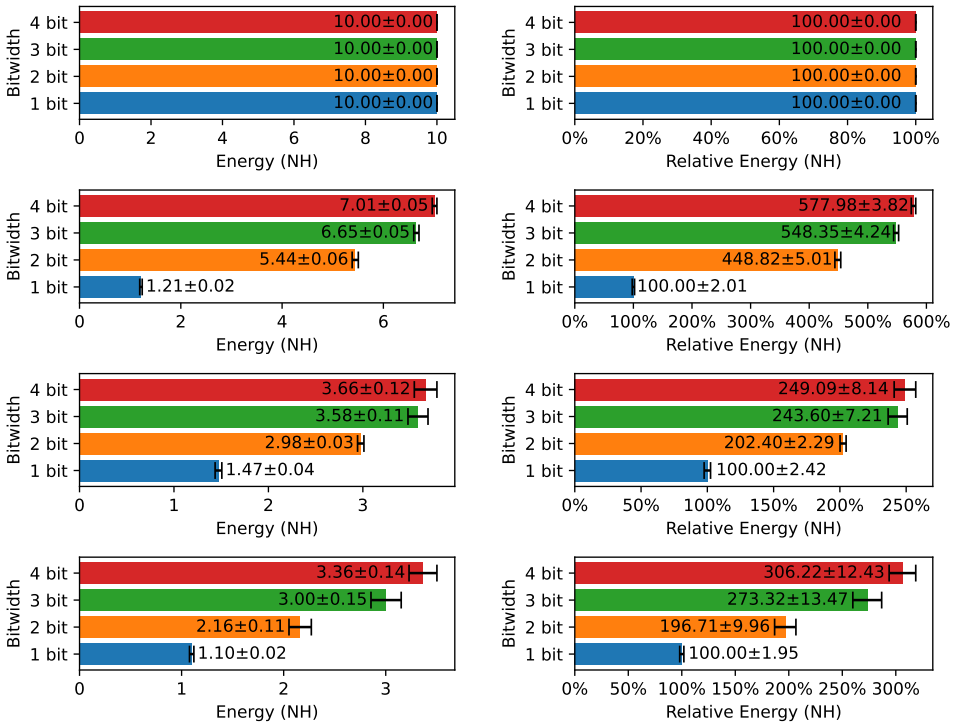


(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.9: Inference Energy Consumption Estimation on Intel Loihi 2

C.2.5 CIFAR-10



(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.10: Inference Energy Consumption Estimation on Intel Loihi 2

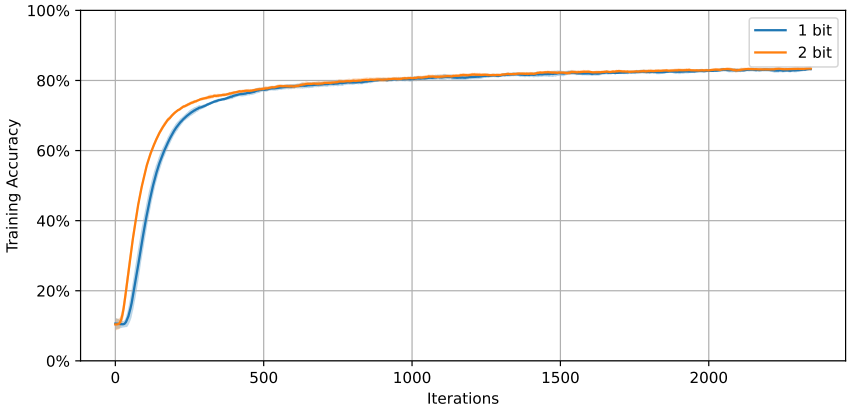
C.3 Trading Accuracy for Energy Consumption

The hyperparameters used in the training process are shown in Table C.1.

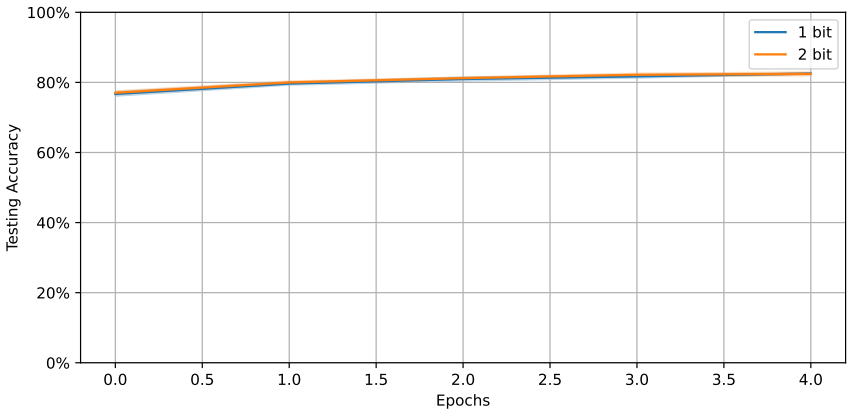
Dataset	Reps	Epochs	LR	Opt	Batch	Time steps
Fashion MNIST	10	5	2e-3	Adam	128	4
CIFAR-10	5	50	1e-5	Adam	128	4

Table C.1: Hyperparameters

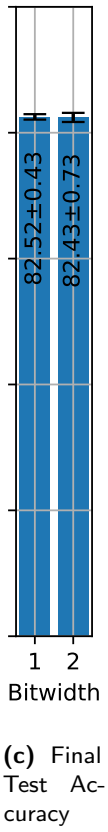
C.3.1 Fashion MNIST



(a) Training Accuracy (smoothed with a window size of 100)

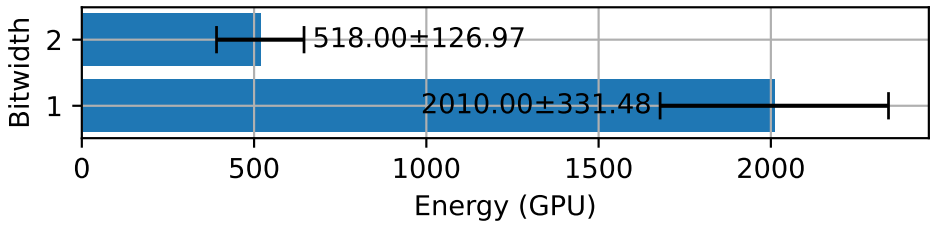


(b) Test Accuracy

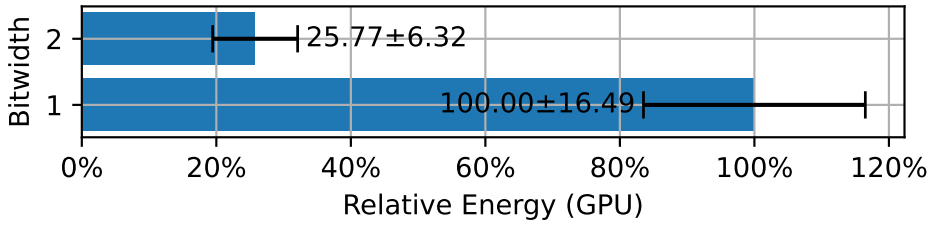


(c) Final Test Accuracy

Figure C.11: Accuracy of 1-bit Spike Train Model with 10 Timesteps and 2-bit Spike Train Model with 4 Timesteps

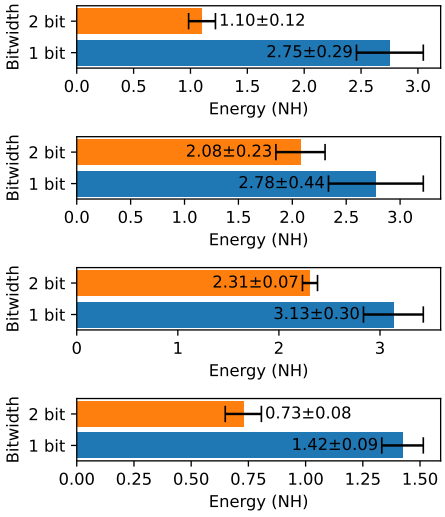


(a) Energy Consumption Estimation, Unit in Constant c

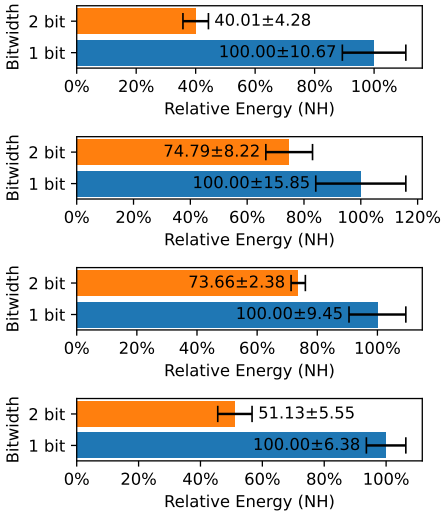


(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.12: Training Energy Consumption Estimation on GPUs for Fashion MNIST Dataset



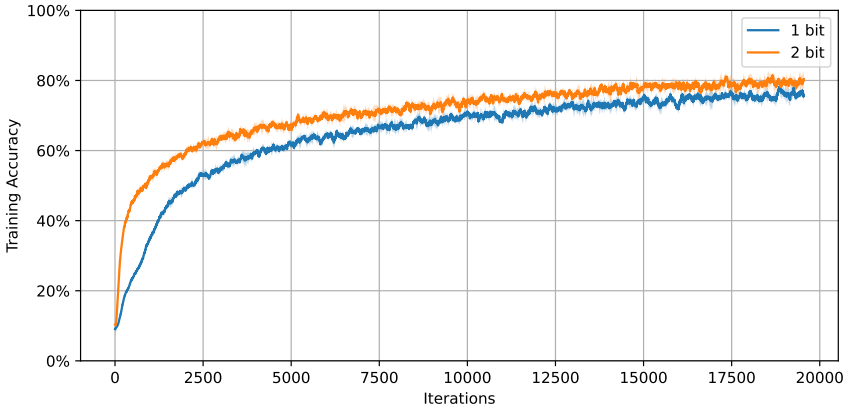
(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$



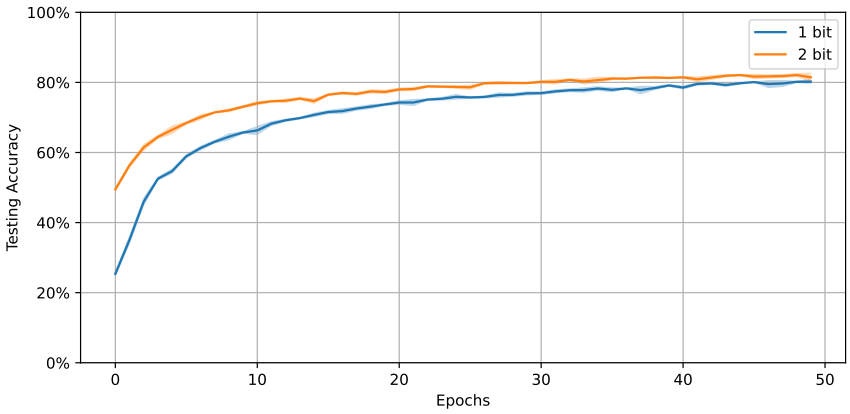
(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.13: Inference Energy Consumption Estimation on Intel Loihi 2

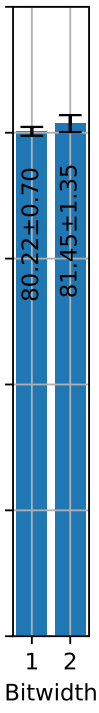
C.3.2 CIFAR-10



(a) Training Accuracy (smoothed with a window size of 100)

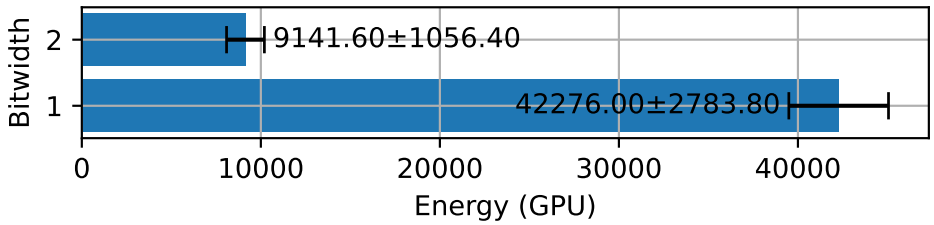


(b) Test Accuracy

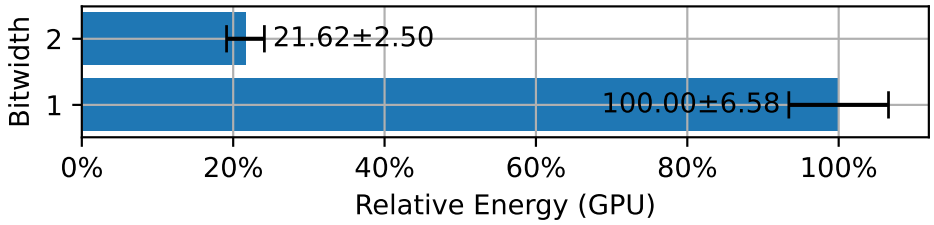


(c) Final Test Accuracy

Figure C.14: Accuracy of 1-bit Spike Train Model with 10 Timesteps and 2-bit Spike Train Model with 4 Timesteps

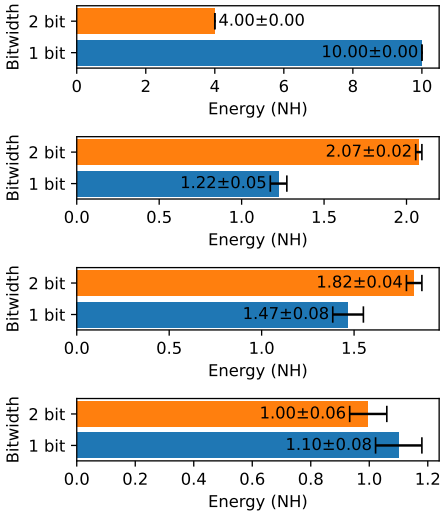


(a) Energy Consumption Estimation, Unit in Constant c

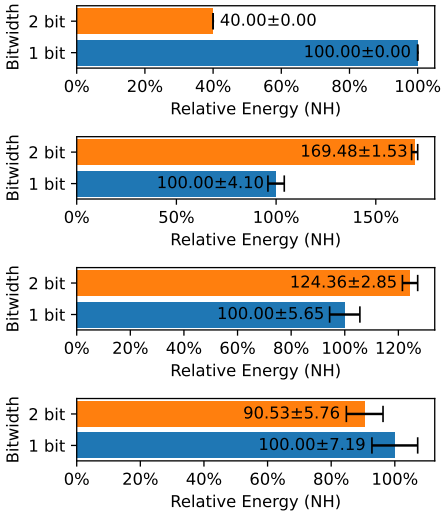


(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.15: Training Energy Consumption Estimation on GPUs for CIFAR-10 Dataset



(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$



(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure C.16: Inference Energy Consumption Estimation on Intel Loihi 2

Bibliography

[1] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[2] Kyoungsu Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37:1311–1314, 06 2004.

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

[4] Alex de Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023.

[5] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[6] Louis Lapique. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J. of Physiol. and Pathology*, 9:620–635, 1907.

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

[8] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13, 2019.

[9] Rui-Jie Zhu, Qihang Zhao, Guoqi Li, and Jason K. Eshraghian. Spikegpt: Generative pre-trained language model with spiking neural networks, 2024.

[10] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[11] Sumit Bam Shrestha and Garrick Orchard. SLAYER: Spike layer error reassignment in time. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1419–1428. Curran Associates, Inc., 2018.

[12] Felix Christian Bauer, Gregor Lenz, Saeid Haghighatshoar, and Sadique Sheik. Exodus: Stable and efficient training of spiking neural networks. *arXiv preprint arXiv:2205.10242*, 2022.

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

[14] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[15] Steven M. Chase and Eric D. Young. First-spike latency information in single neurons increases when referenced to population onset. *Proceedings of the National Academy of Sciences*, 104(12):5175–5180, 2007.

[16] Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40):eadi1480, 2023.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[18] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery.

[19] Gregor Lenz, Kenneth Chaney, Sumit Bam Shrestha, Omar Oubari, Serge Picaud, and Guido Zarrella. Tonic: event-based datasets and transformations., July 2021.

[20] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[23] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[24] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9, 2015.

[25] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7388–7397, 2017.

[26] Rachmad Vidya Wicaksana Putra and Muhammad Shafique. Q-spinn: A framework for quantizing spiking neural networks. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[27] Wenjie Wei, Yu Liang, Ammar Belatreche, Yichen Xiao, Honglin Cao, Zhenbang Ren, Guoqing Wang, Malu Zhang, and Yang Yang. Q-snns: Quantized spiking neural networks. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM '24, page 8441–8450, New York, NY, USA, 2024. Association for Computing Machinery.

[28] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[29] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente B.

Kuang, Rajit Manohar, William P. Risk, Bryan Jackson, and Dharmendra S. Modha. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[30] Ding Chen, Peixi Peng, Tiejun Huang, and Yonghong Tian. Deep reinforcement learning with spiking q-learning, 2024.

[31] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014.

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

[33] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275(5297):213–215, 1997.

[34] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms, 2024.

[35] Yongjun Xiao, Xianlong Tian, Yongqi Ding, Pei He, Mengmeng Jing, and Lin Zuo. Multi-bit mechanism: A novel information transmission paradigm for spiking neural networks, 2024.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].

I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.

Title of paper or thesis:

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s): **First name(s):**

With my signature I confirm the following:
- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date **Signature(s)**

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard