Posit and Floating-Point Based Izhikevich Neuron: A Comparison of Arithmetic

Fernandez-Hart, T.[a,1,*], Knight, James C.[b], Kalganova, T.[a]

[a] *Brunel University, CEDPS, Department of Electronic and Computer Engineering, UB8 3PH, Uxbridge, UK*
[b] *University of Sussex, School of Engineering and Informatics, Brighton, BN1 9QJ,*

Abstract

Reduced precision number formats have become increasingly popular in various fields of computational science, as they offer the potential to enhance energy efficiency, reduce silicon usage, and improve processing speed. However, this is often at the expense of introducing arithmetic errors that can impact the accuracy of a system. The optimal balance must be struck, judiciously choosing a number format using as few bits as possible, while minimising accuracy loss.

In this study, we examine one such format, posit arithmetic as a replacement for floating-point when conducting spiking neuron simulations, specifically using the Izhikevich neuron model. This model is capable of simulating complex neural firing behaviours, 20 of which were originally identified by Izhikevich and are used in this study. We compare the accuracy, spike count, and spike timing of the two arithmetic systems at different bit-depths against a 64-bit floating-point gold-standard. Additionally, we test a rescaled set of Izhikevich equations to mitigate against arithmetic errors by taking advantage of posit arithmetic's tapered accuracy.

Our findings indicate that there is no difference in performance between 32-bit posit, 32-bit floating-point, and our reference standard for 95% of the tested firing types. However, at 16-bit, both arithmetic systems diverge from the 64-bit reference, albeit non-uniformly. For instance, the posit implementation demonstrates an accumulated spike timing error of 0.5ms over a 1000ms simulation compared to 9ms for floating-point – an 18x improvement using posit arithmetic for regular (tonic) spiking. This finding holds particular importance given the prevalence of this particular firing type in specific regions of the brain. Furthermore, when we rescale the neuron equations, this error is eliminated altogether. Hence, our results demonstrate that posit arithmetic is not only a viable replacement for 64-bit floating-point in these simulations, it can do so while using 4× fewer bits. As a Posit Arithmetic Unit has similar area to a Floating Point Unit with the same bit width, this constitutes a significant saving of hardware resources while maintaining full accuracy compared to 64-bit floating-point.

Keywords: floating-point arithmetic, posit arithmetic, spiking neural network, Izhikevich neuron model

1. Introduction

Spiking Neural Networks (SNNs) are often considered the next generation of Artificial Neural Network (ANNs) [40]. They are a closer approximation of biological neural networks than current state-of-the-art ANNs such as convolutional neural networks and operate by simulating the behavior of individual neurons, specifically by modelling their membrane voltages. Unlike ANNs, where neurons are usually continuously active, SNNs exhibit sparse activity in time with neurons only generating brief output 'spikes' when they receives sufficient input. These spikes serve as the means of communication between neurons and facilitate the transfer of information within the SNN. Each spike is considered identical and information is encoded only in the time at which they occur [2]. This sparsity means that SNNs share many of the benefits of their biological counterparts such as low power and noise robustness, and are an active area of research [13] [11]. While not yet mainstream, SNNs have found success in a variety of applications such as object recognition [6], robotic control [5] and even ChatGPT-like large language models [48].

Typically, SNNs use first-order differential equations to describe models of neurons and synapses. Multiple sets of equations are available, varying in their level of complexity and biological correctness. The simplest are the Integrate-and-Fire (IF) family, which are not biologically plausible, but able to mimic network dynamics [46]. The most complex are the Hodgkin-Huxley (HH) model, which are a biophysically accurate neuron model [22]. In the present study, we use the Izhikevich model, which is less complex than the HH, but still able to replicate many neuron firing behaviours and is far more realistic than the IF models [25]. Once a neuron model is chosen for the SNN, a numerical simulation proceeds by tracking the evolution of each neuron's membrane voltage through time. With the Izhikevich model, each neuron has a threshold, which if

*Corresponding author
Email addresses: Tim.Fernandez-Hart@brunel.ac.uk (Fernandez-Hart, T.), J.C.Knight@sussex.ac.uk (Knight, James C.), Tatiana.Kalganova@brunel.ac.uk (Kalganova, T.)

it is crossed, constitutes spike emission, followed by some reset conditions [27].

However, the combination of these complex dynamics and spike-based commmunication leads to challenges running SNN simulations efficiently on a CPU or GPU [31]. Some Application-Specific Integrated Circuit (ASIC) hardware is available such as the SpiNNaker and Loihi neuromorphic systems, but these are research orientated and not widely available [8], [12]. Field Programmable Gate Arrays (FPGA) [47] are another option, but their capacity is limited compared to ASIC, which restricts network size (although they can be linked into multi-board systems, such as Moore et al. [37]). Hence, considerable research has focused on reducing the hardware cost of these networks, while maintaining accuracy in the spike timing and underlying neuron equations. For this purpose, several approaches have been proposed for implementing complex functions, such as CORDIC [21],[18], Look-up tables [4] and piecewise-linear-approximation [3]. More recently, there is growing research interest in reduced precision number formats as a system optimisation technique [30, 36]. The potential benefits to doing so are twofold. Firstly, reducing hardware complexity can free up resources that can be redeployed to increase model complexity. Secondly, reduced bit-depth increases the speed of computation, which in turn, can reduce energy requirements and allow long running dynamics to be simulated.

Posit numbers represent one exciting form of reduced precision number format that promise the accuracy of Floating-Point (FP) while using fewer bits. Additionally, they may be simpler to implement in hardware, requiring less silicon and enabling faster, smaller arithmetic units within processors [15], [7]. But, to the best of our knowledge, only one study has been conducted into the potential use of posit arithmetic for SNN implementations. Silva et al. [43] demonstrated that a single, posit-based Izhikevich neuron was able to replicate the dynamics of 20 different firing types that were originally identified by Izhikevich in his earlier work [27]. However, there are no published studies that explore the impact of reduced precision, posit arithmetic on SNN accuracy and, here we seek to do just this.

We compare the accuracy of equivalently sized posit arithmetic with single precision (32-bit) and half precision (16-bit) FP arithmetic when simulating an Izhikevich neuron while also investigating whether rescaling of the standard Izhikevich equations, to make better use of posit arithmetic's tapered accuracy, is a successful strategy to minimise accuracy loss. The results show that it is possible to simulate tonic firing with 16-bit posit arithmetic while incurring no loss of accuracy, compared to a 64-bit floating-point version. This is an important result given the common usage of this firing type [44], possibly due to its abundance in certain brain areas (90-95% of rat basal ganglia are cells of this type [39, 42]).

Additionally, Chaurasiya et al. [7] showed that hardware usage for a given bit-depth, of a dedicated floating-point unit and posit arithmetic unit are comparable in terms of energy and area. However, they argue that the additional accuracy of posit arithmetic allows for *smaller* posit arithmetic units to be used without sacrificing accuracy. Therefore, our finding of no accuracy loss for 16-bit arithmetic will reduce the overall size of a system, compared to a 64-bit implementation. This saving is likely to be significant. Chaurasiya et al. [7] compared the FPGA utilisation of IEEE-754 compliant adders and multipliers for 32-bit and 16-bit to their posit equivalents. They found that 32-bit floating-point adder used 1049 LUTs, while a 16-bit posit adder only required 391. Similarly, a 32-bit floating-point multiplier required 533 LUTs and 4 DSP slices, whereas the 16-bit posit only needed 218 LUTs and 1 DSP slice. They did not investigate 64-bit arithmetic units. Direct hardware usage comparisons have been reported [10] however, such studies often subset the IEEE-754 standard to reduce the size of the floating-point unit. For example, only implementing a single rounding mode, or not supporting subnormal numbers. Indeed, this approach of sub-setting the full IEEE-754 standard, to reduce the hardware overheads, is gaining some interest in the research community [38, 32, 41].

The rest of this paper is organised as follows. Section 1.2 discusses the two arithmetics and gives an example, illustrating their differences. This includes a short introduction to decimal accuracy, which is a measure of arithmetic precision for a given value. Section 2 outlines the methods used – including all formulas and parameter values for both standard and rescaled equation types – as well as identifying the software emulators and numerical methods used. Section 3 covers the main results of this study as well as a discussion of possible explanations for the differences found. Then, we briefly explain some of the issues with using reduced-precision fixed-point arithmetic to simulate Izhikevich neurons in Section 3.4. Finally, Section 4 proposes the main conclusions and suggests future research directions.

1.1. Contributions to Science

- We not only show that posit arithmetic is capable of matching floating-point at reduced precision, but we quantify the arithmetic errors for both number systems for all 20 firing patterns of the Izhikevich neuron model. Moreover, we show when posits outperform floating-point numbers.
- We demonstrate that rescaling the standard Izhikevich equations is a powerful technique for mitigating against arithmetic errors when using reduced precision floating-point and posit number formats.
- Finally, we show that regular spiking (Tonic Spiking) can be simulated using 16-bit posit arithmetic with *no loss of accuracy* compared to 64-bit floating point - a potential 75% saving in hardware resources.

1.2. Background

Currently, the predominant format for representing real numbers within a computer is Floating-Point (FP), which is ratified in the IEEE-754 standard [1]. As shown in Figure 1, floating point numbers represent numbers using sign, exponent and fraction bits. This arrangement has a large dynamic range and it is the most widely used number format in scientific computing. However, it has several well-known drawbacks [4]. For instance, floating-point arithmetic is neither associative nor commutative making it inconsistent with fundamental mathematical laws. Additionally, it has two representations of zero, permits subnormal numbers, has multiple rounding modes and reserves a great many bit patterns for Not-a-Number (NaN). These increase the format's hardware cost, due to the need to verify all these special conditions during run time.

Posit arithmetic is designed to be a hardware friendly, drop-in replacement for FP although, as it was only introduced in 2017, it is still in its infancy [15]. As shown in Figure 1, posits have sign and fraction bits like FP. However, unlike FP, the fraction is scaled by two values – the exponent and the regime – rather than just the exponent as in FP. The regime bits are unique to posits, and act like a super-exponent, scaling the exponent and thus the fraction. From a hardware perspective, the main drawback of posit arithmetic is the dynamic size of the regime bits, which alter the location of the exponent and fraction bits. Thus, posit decoding entails extra overhead encoding and decoding values compared to FP, required to determine the boundaries of each of these sections, before any further operations can be performed. However, even with this overhead, hardware implementations of posit arithmetic units should still be more efficient than IEEE-754 compliant FP units of the same bit depth due to the intricacies of the IEEE-754 standard. For example, the posit standard requires only a single rounding mode compared to five for FP. Indeed, given these implementation complexities and the fact that a 64-bit, IEEE-754 compliant FP arithmetic unit (FPU) entail can occupy up to 60-70% of the area of a CPU, hardware designers are moving away from full standard compliant FPU designs as part of a move toward approximate computing [7].

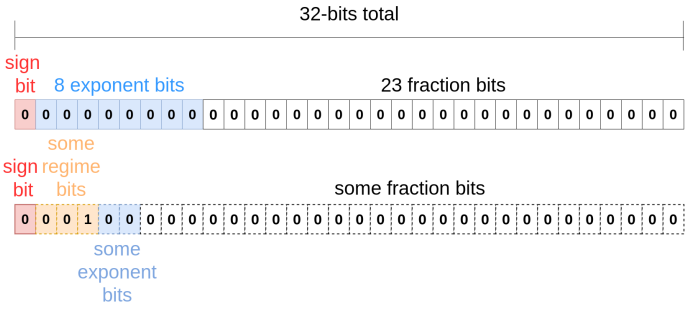


Figure 1: A comparison of 32-bit floating-point (top) and 32-bit posit (bottom) number structures. 32-bit floating-point structure as defined in the IEEE-754 standard. 1 sign bit, 8 exponent bits and 23 fraction bits. A 16-bit posit (posit16) is defined as 1 sign bit, at least 2 regime bits, 2 exponent bits and the remaining bits represent the fraction. The dashed lines on (b) represent dynamically positioned bits.

A posit format is declared with two numbers posit$\langle n, \text{es}\rangle$ where n is the total number of bits and es, the maximum number of those bits used for the exponent. The first bit of a posit is always a sign bit, which is immediately followed by the regime bits. The regime bits encode the value r in a run of zeros, or ones, terminated by the opposite bit. A run of ones encodes a positive number, and a run of zeros a negative number, as defined by Equation 2 [14]. Where k is the number of ones or zeros before the terminating bit. The regime bits are not present in the floating-point standard and are of variable length. In the extreme case, they can expand to fill the whole word, leaving no space for the exponent or any part of the fraction. Following the regime bits are the exponent bits. Unlike floating-point, the exponent bits do not have a bias. Lastly, we have the fraction bits. These contain an implicit one and function in the same way as in floating point. A number p is decoded using Equation 1. The value useed is calculated by useed $= 2^{2^{es}}$ (see [15] for more explanation).

$$p = (-1)^s \text{useed}^r 2^e \left(1 + \frac{f}{2^{fs}}\right) \tag{1}$$

$$r = \begin{cases} -k & \text{if} R_0 = 0 \\ (k-1) & \text{if} R_0 = 1 \end{cases} \tag{2}$$

Importantly, posits do not underflow or overflow. Instead, they exhibit tapered precision, centered around 0 and greatest in the interval (-1, 1). This has implications for their accuracy. For example, trying to represent Euler's number $e = 2.7182818284590452354$ with 16-bit Floating-Point (FP16) and 16-bit posit (posit16) results in the bit patterns shown in Figure 2. The benefit of dynamically positioned bits means that, in a particular range, the posit's regime bits are at their shortest (2 bits minimum), when combined with fewer exponent bits, leaves more bits to represent the fraction. These additional fraction bits can be seen to improve the accuracy of the representation.

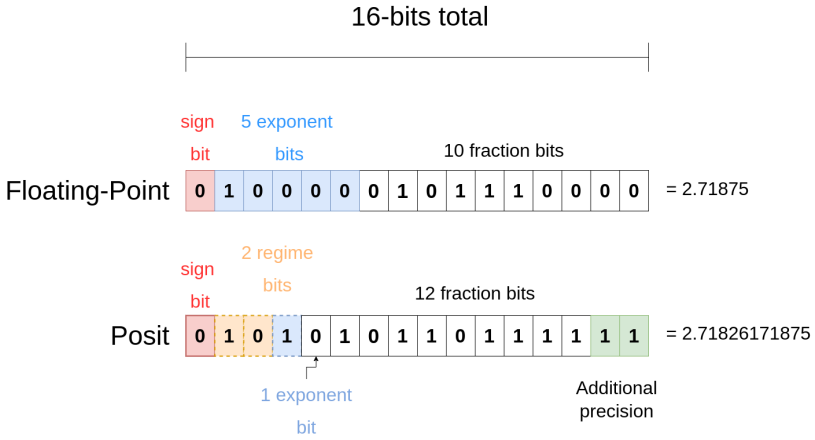


Figure 2: 16-bit example of Euler's number represented in FP16 and Posit16 arithmetic. The additional 2 fractional bits have been highlighted in green.

Decoding Figure 2 into actual values gives us a posit16 value of 2.71826171875 (Equation 4).

$$p = (-1)^0 \times 4^0 \times 2^1 \times \left(1 + \frac{1471}{2^{12}}\right) \tag{3}$$

$$= 2.71826171875 \tag{4}$$

If we compare this result with the FP16 version of e, which has 2 fewer fraction bits, we can see the effect on the accuracy of its representation. FP16 achieves $e = 2.71875$.

$$fp = (-1)^s \times 2^{e-bias} \times \left(1 + \frac{fraction}{2^{10}}\right) \tag{5}$$

$$fp = (-1)^0 \times 2^{16-15} \times \left(1 + \frac{368}{2^{10}}\right) \tag{6}$$

$$= 2.71875 \tag{7}$$

One method to quantify this difference is to use decimal accuracy (Equation 8), which quantifies the number of correct decimal places following rounding [15, 30]. Using this on the Euler's number example gives a decimal precision of 5.49 for posit16 and 4.13 for FP16, illustrating the advantage of posit's tapered accuracy.

$$\text{decimal accuracy} = -\log_{10}\left|\log_{10}\left(\frac{x_{\text{repr}}}{x_{\text{exact}}}\right)\right| \tag{8}$$

In addition to tapered accuracy, the posit numbering scheme has several advantages over FP. Firstly, only two exception values exist, zero and NaR (Not-a-Real, equivalent to Not-a-Number and representative of $\pm\infty$). This not only reduces costly condition checking but additionally, it liberates bit patterns to represent values, which in turn, gives posit arithmetic a higher precision for a given number of bits compared to FP. Secondly, posits use a single rounding scheme. Hence, posit circuitry should be more compact than a current dedicated FP engine and at least as accurate. However, it is unclear what effect these differences will have on SNN simulations.

2. Methods

We used single-precision (FP32) and half-precision (FP16) floating-point numbers, along with the posit$\langle 32, 2\rangle$ (posit32) and posit$\langle 16, 1\rangle$ (posit16) formats formally defined by the Posit Working Group [14] to run simulations of an Izhikevich neuron over 1000ms using both standard equations and rescaled equations. Following the approach employed by several previous studies [9, 30], we used a software implementation to study the errors in numeric simulations of reduced precision FP and posit arithmetic. Although emulation does not give a measure of speed, it does inform us about accuracy of the arithmetic. To compare the membrane voltage of our simulations against a 64-bit floating point baseline, we measure the Normalised Root-Mean-Square Error (NRMSE) of each test. Spike counts and their accumulated timing error were also recorded.

2.1. Model Definitions

2.1.1. Standard Neuron Model

In 2003, Izhikevich presented a neuron model capable of complex spiking patterns, similar to those found in nature (e.g. intrinsic bursting, chattering, low-threshold spiking, fast spiking) but with a reduced computational load in comparison to other models [27]. It consists of a pair of non-linear differential equations: Equation 9 models to evolution of the membrane voltage and equation 10 models a recovery variable.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{9}$$

$$u' = a(bv - u) \tag{10}$$

Once the membrane voltage crosses a 'threshold voltage', typically 30mV for the Izhikevich model, a spike is emitted and predefined reset conditions are applied:

$$v \leftarrow c \tag{11}$$

$$u \leftarrow u + d \tag{12}$$

The behaviour of the model is controlled by four dimensionless variables a, b, c, d which correspond to different model properties [25] and are typically set to the values listed in Table 1 in order to reproduce one of the 20 identified behaviours (See [27]). Three firing patterns (Class 1, Integrator and Accommodation) also require altering Equations 9 and 10, see the authors website for full implementation details [26]. Given its phenomenological fidelity with nature, coupled with its computational efficiency the Izhikevich model is widely used in many SNN implementations and has been used to compare arithmetics in previous studies [23, 24]. Hence, it was also selected as the neuron type in this study.

2.1.2. Rescaled Neuron Model

In addition to the standard Izhikevich model, we developed a rescaled version so that the standard operating range of the state variables would fit within the region where posits are most accurate. This transforms Equation 9 into Equation 13. Equations 11 and 12 which describe the reset behaviour are unchanged but the c and d parameters as well as the input current, spike threshold and initial value of u and v are all rescaled by a factor of 0.01.

$$v' = 4v^2 + 5v + 1.4 - u + I \tag{13}$$

2.2. Testing Procedure

We compared FP32, FP16, posit32 and posit16 against a 64-bit floating-point (FP64) ground-truth for all 20 firing types outlined above, using both standard and rescaled equations. Although several posit hardware implementations have been proposed [35, 28], the hardware is not yet widely available. However, several open-source software simulation libraries are available with the SoftPosit library being the most prominent. Here, we use this library to test the accuracy of posit arithmetic and compare it against other varieties [10, 34] and the Berkeley SoftFloat library [16] to simulate FP32 and FP16 formats. We ran our simulations in Python 3.8, on a 64-bit machine using wrappers for SoftPosit and SoftFloat (developed by Leong [33] and Leong [34] respectively) and used the native floating-point type for the reference FP64 simulations. Simulations were run for 1000ms using the Forward Euler method of numerical integration with time step sizes laid out in Table 1.

2.3. Data Analysis

For each test, we recorded the voltage (mV) every time step and, if the spike threshold was crossed, the time step at which this occurred (ms). These recorded values were used to calculate: the membrane voltage error, spike count error, and where the spike count was the same, the accumulated spike timing error.

2.3.1. Membrane Voltage Error

In line with similar studies [19, 17], membrane voltage error was quantified using the Normalised Root Mean Square Error (NRMSE):

$$\text{RMSE} = \sqrt{\frac{\Sigma_{t=1}^{n}(v_{\text{aut}}(t) - v_{\text{fp64}}(t))^2}{n}} \tag{14}$$

$$\text{NRMSE} = \frac{\text{RMSE}}{(v_{\max} - v_{\min})} \tag{15}$$

Where v_{aut} is the membrane voltage of the *arithmetic under test*, and v_{fp64} is the membrane voltage simulated using 64-bit floating-point arithmetic at time t. NRMSE quantifies the error between the FP64 voltage reference and the arithmetic under test at each time step during the each simulation. Normalising the error allows errors to be directly compared between standardised and rescaled equation types.

2.3.2. Spike Count and Timing

In order to assess the effect of arithmetic error on spike count and their timing, we also counted the number of spikes and their absolute accumulated lag/lead times. First, we determined whether each test produced the expected number of spikes (n_{spike}). If it did, we then looked at the timing of each spike (S_i^{aut}), relative to the reference FP64 simulation (S_i^{fp64}):

$$T = \sum_{i}^{n_{\text{spike}}} |S_i^{\text{fp64}} - S_i^{\text{aut}}| \tag{16}$$

Where T is the sum of absolute timing difference per spike, across the whole simulation. This method differs slightly from Hopkins et al. [24], in that the final reported result is the accumulated lead/lag, as opposed to the difference for each spike. The main reason for this is brevity as we tested all 20 Izhikevich neuron behaviours whereas Hopkins et al. [24] only tested a few firing types. Moreover, accumulating the timing errors will still allow a comparison between experiments.

3. Results and Discussion

3.1. Membrane Voltage Error

First, we present the NRMSE results for the different arithmetics, bit depths and equation types in Table 3; Figure 3 and Figure 4 show the 32-bit and 16-bit results respectively. Aside from the 'Class 2' Izhikevich neurons – which were equally problematic at all precisions and equation types – the NRMSE was far lower for 32-bit arithmetic compared to 16-bit.

Considering the standard Izhikevich equations, Posit32 had a lower NRMSE in every case with the exception of 'Class 2'. Since the 'Class 2' error was much greater than the other errors and it was lower for FP32 it skewed the

Firing Type	a	b	c	d	dt	initial u	initial v
Tonic Spiking	0.02	0.2	-65.0	6.0	0.25	$b \times v$	-70
Phasic Spiking	0.02	0.25	-65.0	6.0	0.25	$b \times v$	-64
Tonic Bursting	0.02	0.2	-50.0	2.0	0.25	$b \times v$	-70
Phasic Bursting	0.02	0.25	-55.0	0.05	0.2	$b \times v$	-64
Mixed Mode	0.02	0.2	-55.0	4.0	0.25	$b \times v$	-70
SFA	0.01	0.2	-65.0	8.0	0.25	$b \times v$	-70
Class 1	0.02	-0.1	-55.0	6.0	0.25	$b \times v$	-60
Class 2	0.2	0.26	-65.0	0.0	0.25	$b \times v$	-64
Spike Latency	0.02	0.2	-65.0	6.0	0.2	$b \times v$	-70
Subthreshold Oscillation	0.05	0.26	-60.0	0.0	0.25	$b \times v$	-62
Resonator	0.1	0.26	-60.0	-1.0	0.25	$b \times v$	-62
Integrator	0.02	-0.1	-55.0	6.0	0.25	$b \times v$	-60
Rebound Spike	0.03	0.25	-60.0	4.0	0.2	$b \times v$	-64
Rebound Burst	0.03	0.25	-52.0	0.0	0.2	$b \times v$	-64
Threshold Variability	0.03	0.25	-60.0	4.0	0.25	$b \times v$	-64
Bistability	0.1	0.26	-60.0	0.0	0.25	$b \times v$	-61
DAP	1.0	0.2	-60.0	-21.0	0.1	$b \times v$	-70
Accomodation	0.02	1.0	-55.0	4.0	0.5	-16	-65
Inhibition Induced Spiking	-0.02	-1.0	-60.0	8.0	0.5	$b \times v$	-63.8
Inhibition Induced Bursting	-0.02	-1.0	-45.0	0.0	0.5	$b \times v$	-63.8

Table 1: Izhikevich parameters and initial conditions to produce 20 different firing types. Additionally, for firing types Class 1 and Integrator, Equation 9 becomes $v' = 0.04v^2 + 4.1v + 108 - u + I$ and for Accommodation, Equation 10 becomes $u' = a(b(v + 65))$ [27].

overall mean which was 0.0115 for FP32 and 0.0119 for posit32. Indeed, with 'Class 2' removed from the calculation, the overall mean for FP32 was 2.58×10^{-5} and 3.97×10^{-6} for posit32, which is an error reduction of over 6x. A broadly similar pattern was noted for the rescaled equations. Posit32 had a lower NRMSE in 18 of the 20 firing types. 'Bistability' and 'Class 2', both had lower errors in FP32. 'Bistability' did not demonstrate the expected cessation of firing on its second input spike. This behaviour is very sensitive to timing, and posit32 was not able to demonstrate it. However, with these outlier removed the overall mean NRMSE for FP32 was 2.32×10^{-5} and 1.72×10^{-6} for posit32, which is an error reduction of $13.5\times$ in the firing types with expected dynamics.

The standard Izhikevich equations for all firing types need to represent a minimum value of -81.8 and a maximum of 30. Since FP32 is a fixed format, it always uses 23 fractional bits to express these values whereas posit32 will use 27 fractional bits for values around 30, and 23 fractional bits to represent -81.8. Hence, when using the standard equations, posit32 has *at least* the same number of fractional bits as FP32 resulting in additional precision across most of the numerical range encountered in these simulations. Table 2 gives a summary of the number of fractional bits, available for each arithmetic at these values and their decimal accuracy.

Next we consider 16-bit arithmetic. Figure 4 clearly shows that some firing types such as 'Class 1' (which has an NRMSE close to 0.2 for all arithmetics) accrue more error than others such as 'Spike Latency', 'Subthreshold' and 'Resonator' whose NRMSE is below 0.05. This pattern broadly correlates with the number of spikes emitted in a simulation, shown by the dotted black line in Figure 4. However, there were exceptions to this. Most notably, the error was far greater for FP16 with 'Rebound Burst' (standard equations). This was due to the neuron continuing to fire regularly following the expected burst of spikes. This is in contrast to the posit16 test which failed to fire the expected burst at all.

Using the standard equations, posit16 had a lower NRMSE in 11 of the 20 firing types. This is reflected in the mean error which is slightly lower for posit16 (0.0724) than FP16 (0.0782). Unlike the 32-bit tests, there were no experiments where one arithmetic or firing type was orders of magnitude different from the rest, requiring additional consideration. Rescaling improved the error for FP16 in only 6 firing types and the mean error increased slightly. This in contrast to the posit16 results, which showed an improvement in 13 firing types compared to the standard equations, with a reduction in the mean error. This clearly demonstrates the use of rescaling to improve simulation accuracy when using reduced precision posit arithmetic.

One possible explanation for this is the increase in the number of available fractional bits following rescaling. For example, posit16 had 10 fractional bits to represent the value 30, the same as FP16. However, when we use the rescaled equations, the value 30 becomes 0.3 where posit16 uses 12 fractional bits. Again, it is likely that it is these additional fractional bits that give posit16 the advantage over FP16. A similar situation can be seen for the lower bound number -81.8. At this value, posit16 has only 9 fractional bits compared to 10 used in FP16. However, in the rescaled equations, -81.8 becomes -0.818 and posit16 provides 12 fractional bits compared to FP16's 10.

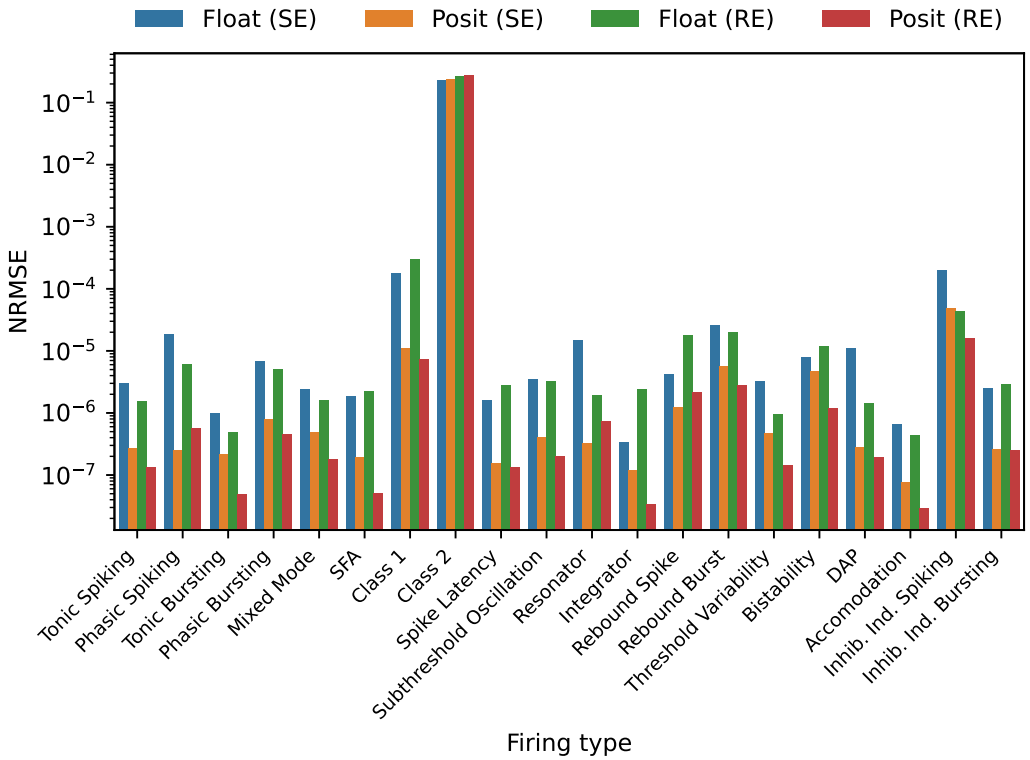


Figure 3: 32-bit floating-point and posit arithmetics using both the Standard Equations (SE) and Rescaled Equations (RE). Where Inhib. Ind. Spiking is Inhibition Induced Spiking and Inhib. Ind. Bursting is Inhibition Induced Bursting. NRMSE is displayed on a log scale to account for the magnitude of the 'Class 2' error.

		Number of Bits for Fraction		Decimal Accuracy		Number of Bits for Fraction		Decimal Accuracy	
Equation Type	**Value**	**FP32**	**Posit32**	**FP32**	**Posit32**	**FP16**	**Posit16**	**FP16**	**Posit16**
Standard	30	23	26	Exact	Exact	10	10	Exact	Exact
	0	23	0 (exception)	Exact	Exact	10	0 (exception)	Exact	Exact
	-81.8	23	26	7.79	8.99	10	9	4.18	3.58
Rescaled	0.3	23	27	7.76	8.97	10	12	4.15	4.75
	0	23	0 (exception)	Exact	Exact	10	0 (exception)	Exact	Exact
	-0.818	23	27	8.01	10.80	10	12	4.16	5.44

Table 2: A table showing the number of bits used to represent the fraction for each arithmetic type and equation. The values were chosen as the minimum and maximum values seen across all firing types. Decimal accuracy shows the correct number of decimal places after rounding for each value.

As well as NRMSE, we can also calculate decimal precision for each arithmetic and equation type. Using the standard equations the threshold value is 30, which is exactly representable in all arithmetics and bit depths. However, in the rescaled equations, the threshold becomes 0.3 where FP32 has 23 fractional bits and a decimal precision of 7.76 digits whereas posit32 has 26 fractional bits and a decimal precision of 8.97, resulting in more than one correct additional decimal place. Using reduced 16-bit arithmetic, FP16 has 10 bits for the fraction and achieves a decimal precision of 4.15 whereas posit16 has 12 fractional bits and a decimal accuracy of 4.75. Tamura et al. [45] demonstrated that small changes can have a large impact on the Izhikevich model. Their bifurcation analysis of the Izhikevich system showed that, a change in the parameter b by as little as 0.07 can change a stable system into a chaotic one.

Table 2 compares the number of bits available for each test to represent the fraction for three values. These values were the minimum and maximum seen over all 20 tests for each equation type, plus zero. It shows that when using 32-bits, posit arithmetic has 3 more bits to represent each fraction. Therefore, giving it a higher accuracy. However, when using 16-bits, FP16 and posit16 both have 10 bits to represent the value 30 but posit16 has only 9 bits to represent -81.6, compared to 10 bits for FP16. This could account for why fewer posit tests could reproduce the correct number of spikes in 16-bit tests using the standard equa-

tions. However, overall the NRMSE was very similar to FP. In contrast, when comparing the fraction bits available for the rescaled values, we can see that posit32 has 4 more bits of precision than FP32 and posit16 has 2 more bits than FP16. Thus posit arithmetic always has a higher accuracy when using the rescaled equations. Indeed, rescaling the equations reduced the NRMSE of posit16-based simulations in 13 out of the 20 firing types – dramatically in some cases. This is in contrast to the FP16 results, where only 5 firing types had an improved NRMSE under the rescaled equations – a likely consequence of almost constant decimal accuracy. 'Tonic Spiking' in particular appears to benefit from using the rescaled equations with posit16 arithmetic.

To illustrate this disparity, Figure 7 shows the FP16 simulation of 'Tonic spiking' using standard equations. If we compare this to the posit16 simulation shown in Figure 10, then the advantages of this arithmetic become apparent. If we again focus on the final 50ms, while there is a slight error, it is much reduced compared to the error at the same point when using floating-point arithmetic (Figure 9).

Finally, as shown in Figure 11, this error reduces to effectively zero for the posit16 simulation when we use the rescaled equations – with only an small difference in voltage amplitude separating the two simulations. However, as shown in Figure 11a, rescaling does not reduce the FP16 error.

3.2. Class 2

Figure 12 shows the 'Class 2' firing type using FP32 and standard equations. Izhikevich defines 'Class 2' as neuron types which are either quiescent or have a relatively large firing rate [27]. Although Class 2 has the greatest NRMSE, it demonstrates normal firing, and the high NRMSE are likely to be related to the high spike count, each of which lags or leads the FP64 reference. However, if we consider the spike count, Class 2 is very similar to the FP64 reference.

3.3. Spike Count and Timing

In Table 4 we show the spike count and, if we first consider 32-bit arithmetic, we can see that when using standard equations, posit32 deviated in spike count from the reference only once for the 'Class 2' firing type. FP32 did not have this error and fired the correct number of times. While this change in spike number should be a cause for concern and we will discuss it in the next section, we should also consider spike timing. In table 5 we show the accumulated spike timing errors and, when using 32-bit arithmetic, this shows there were no spike timing errors for any firing type, again with the exception of Class 2. Similarly to other results, the situation was different when 16-bit arithmetic was used. When using standard equations, posit16 deviated from the reference nine times whereas, FP16 deviated only four times. This often meant that the expected dynamics were not achieved in that test. For example, 'Rebound Burst' should fire 11 spikes following a brief inhibitory input. FP16 fired a burst, but then continued firing afterwards meaning that 114 spikes were emitted over the 1000ms. This is in contrast to posit16 which did not fire at all. Hence, in this case, both arithmetics failed to produce the expected burst of spikes. When the equations were rescaled, posit16 improved, deviating from the FP64 only seven times whereas FP16 got worse – deviated 8 times from FP64. Furthermore, with rescaled equations, posit16 was the only 16-bit arithmetic capable of reproducing the expected dynamics of the 'Threshold Variability' firing type. Table 5 shows the mean accumulated timing error. When using 16-bit arithmetic, posit16 had a similar mean value to FP16 across the reproduced firing types when using the standard equations, but a much lower mean error when using the rescaled equations.

3.4. Fixed-Point Arithmetic

While floating-point and posit arithmetic have been investigated in this paper, fixed-point is another potential arithmetic option. Fixed-point arithmetic is much simpler and faster than either posit or floating-point and easier to implement. However, if care is not taken it is highly sensitive to potentially catastrophic underflow and overflow errors. Although both Hopkins et al. [24] and Jin et al. [29] were able to achieve a 16-bit fixed-point representation of an Izhikevich neuron, this was only managed after employing multiple error reducing strategies which make their implementation incompatible with this work.

Hopkins et al. [24] used higher order ODE solvers namely, RK2, RK3 and Chan-Tsai. While these are more accurate, they entail a higher latency and computational overhead than the much simpler Forward Euler used here. Moreover, they chose only a subset of neuron firing types and a smaller time step. Using smaller time-steps generally improves the stability and accuracy of a solution when using numerical methods of integration. Indeed, the accuracy typically increases monotonically as step size reduces, but this not only increase the conputational load it can also interfere with the required neuron behaviour. For example, some firing types are very sensitive to input spike timing such as bistability; If there is too much lag/lead error, bistability doesn't stop firing on the second input, as it should. This behavioural dependence on the timestep size has been previously described in Heidarpur et al. [20] and Skocik and Long [44] and also explains why Hopkins et al. [24] and Jin et al. [29] only investigated a subset of firing types. Hopkins et al. [24] also used mixed precision throughout, opting for 32-bit fixed-point numbers for all constants and pre-computed several of these to control any rounding error. They admit that even after using all these strategies, using round-to-nearest rounding scheme, did not produce any spiking behaviour at all. Indeed, to reduce their errors to an acceptable level they had to employ stochastic-rounding. Which, although relatively sim-

Firing Type	Standard - NRMSE				Rescaled - NRMSE			
	32-bit		16-bit		32-bit rescaled		16-bit rescaled	
	Float	Posit	Float	Posit	Float	Posit	Float	Posit
Tonic Spiking	3.01E-06	2.73E-07	1.62E-01	1.04E-01	1.55E-06	1.34E-07	1.64E-01	7.46E-03
Phasic Spiking	1.84E-05	2.51E-07	4.05E-02	3.20E-02	6.00E-06	5.76E-07	3.36E-02	3.50E-02
Tonic Bursting	9.69E-07	2.13E-07	1.32E-02	8.05E-02	4.93E-07	4.82E-08	1.59E-01	3.42E-03
Phasic Bursting	6.74E-06	7.80E-07	5.68E-02	5.24E-02	5.04E-06	4.53E-07	9.42E-02	9.20E-02
Mixed Mode	2.43E-06	4.91E-07	9.16E-02	7.46E-02	1.58E-06	1.76E-07	1.22E-01	1.25E-01
SFA	1.87E-06	1.95E-07	1.46E-01	1.47E-01	2.21E-06	5.01E-08	7.96E-02	1.65E-01
Class 1	1.79E-04	1.11E-05	2.03E-01	1.86E-01	3.03E-04	7.22E-06	2.03E-01	1.81E-01
Class 2	2.30E-01	2.39E-01	2.79E-01	2.47E-01	2.69E-01	2.79E-01	2.53E-01	2.88E-01
Spike Latency	1.62E-06	1.55E-07	2.03E-02	1.54E-02	2.77E-06	1.33E-07	2.31E-02	1.11E-02
Subthreshold Oscillation	3.49E-06	4.08E-07	2.29E-02	4.26E-02	3.21E-06	1.97E-07	2.79E-02	1.97E-02
Resonator	1.47E-05	3.25E-07	3.33E-02	4.50E-02	1.93E-06	7.37E-07	4.18E-02	2.06E-02
Integrator	3.36E-07	1.20E-07	2.27E-02	1.88E-02	2.42E-06	3.32E-08	2.42E-02	1.45E-02
Rebound Spike	4.22E-06	1.25E-06	2.96E-02	2.30E-02	1.79E-05	2.13E-06	4.80E-02	2.14E-02
Rebound Burst	2.58E-05	5.59E-06	1.82E-01	5.82E-02	1.98E-05	2.80E-06	6.46E-02	5.83E-02
Threshold Variability	3.26E-06	4.69E-07	2.51E-02	4.24E-02	9.53E-07	1.42E-07	4.62E-02	2.85E-02
Bistability	7.96E-06	4.68E-06	1.13E-01	5.18E-02	1.20E-05	1.12E-01	1.04E-01	1.02E-01
DAP	1.08E-05	2.79E-07	6.72E-03	2.86E-02	1.43E-06	1.96E-07	1.25E-02	1.93E-02
Accomodation	6.60E-07	7.67E-08	6.18E-03	7.63E-03	4.38E-07	2.90E-08	8.42E-03	3.52E-03
Inhibition Induced Spiking	2.01E-04	4.84E-05	4.11E-02	9.98E-02	4.41E-05	1.57E-05	5.00E-02	3.58E-02
Inhibition Induced Bursting	2.52E-06	2.55E-07	7.09E-02	9.06E-02	2.86E-06	2.46E-07	6.02E-02	5.98E-02
Mean	1.15E-02	1.19E-02	7.82E-02	7.24E-02	1.35E-02	1.95E-02	8.10E-02	6.46E-02

Table 3: NRMSE for each firing pattern, arithmetic and equation type.

Firing Type	Ref	Standard Equation				Ref	Rescaled Equations			
	64-bit	32-bit		16-bit		64-bit	32-bit		16-bit	
	Float	Float	Posit	Float	Posit	Float	Float	Posit	Float	Posit
Tonic Spiking	38	=	=	=	=	38	=	=	=	=
Phasic Spiking	1	=	=	=	**-1**	1	=	=	=	=
Tonic Bursting	124	=	=	=	=	124	=	=	**-2**	=
Phasic Bursting	7	=	=	=	**-7**	7	=	=	**-1**	**-1**
Mixed Mode	33	=	=	=	=	33	=	=	=	**-1**
SFA	38	=	=	=	=	38	=	=	=	=
Class 1	115	=	=	=	=	115	=	=	=	=
Class 2	166	=	**+1**	**+1**	**+2**	167	=	=	**+1**	**-1**
Spike Latency	1	=	=	=	=	1	=	=	=	=
Subthreshold Oscillation	1	=	=	=	=	1	=	=	=	=
Resonator	1	=	=	=	**-1**	1	=	=	**+1**	=
Integrator	1	=	=	=	=	1	=	=	=	=
Rebound Spike	1	=	=	=	**-1**	1	=	=	**-1**	**-1**
Rebound Burst	11	=	=	**+103**	**-11**	11	=	=	**-11**	**-11**
Threshold Variability	1	=	=	**-1**	**-1**	1	=	=	**-1**	=
Bistability	5	=	=	**+19**	**-3**	5	=	=	**+19**	**+21**
DAP	1	=	=	=	**+2**	1	=	=	=	**+1**
Accomodation	1	=	=	=	=	1	=	=	=	=
Inhibition Induced Spiking	4	=	=	=	=	4	=	=	=	=
Inhibition Induced Bursting	16	=	=	=	=	16	=	=	=	=

Table 4: Spike count deviations over 1000ms for each firing type, equation type and arithmetic. '=' signifies no difference from the 64-bit reference.

Firing Type	Standard Equations Accumulated Spike Timing Error (ms)				Rescaled Equations Accumulated Spike Timing Error (ms)			
	32-bit		16-bit		32-bit		16-bit	
	Float	Posit	Float	Posit	Float	Posit	Float	Posit
Tonic Spiking	0.00	0.00	166.50	18.50	0.00	0.00	166.50	0.00
Phasic Spiking	0.00	0.00	2.50	-	0.00	0.00	0.75	2.50
Tonic Bursting	0.00	0.00	0.00	14.25	0.00	0.00	-	0.00
Phasic Bursting	0.00	0.00	11.80	**-**	0.00	0.00	-	-
Mixed Mode	0.00	0.00	21.75	11.00	0.00	0.00	116.25	-
SFA	0.00	0.00	132.25	140.25	0.00	0.00	8.50	140.25
Class 1	0.00	0.00	106.50	103.0	0.00	0.00	163.25	69.25
Class 2	31.00	-	**-**	**-**	257.50	331.50	-	-
Spike Latency	0.00	0.00	0.19	0.00	0.00	0.00	0.20	0.00
Subthreshold Oscillation	0.00	0.00	0.25	0.00	0.00	0.00	0.25	0.25
Resonator	0.00	0.00	5.00	-	0.00	0.00	-	0.50
Integrator	0.00	0.00	0.75	0.50	0.00	0.00	0.75	0.25
Rebound Spike	0.00	0.00	6.20	-	0.00	0.00	-	-
Rebound Burst	0.00	0.00	**-**	**-**	0.00	0.00	-	-
Threshold Variability	0.00	0.00	-	-	0.00	0.00	-	5.00
Bistability	0.00	0.00	-	**-**	0.00	0.00	-	-
DAP	0.00	0.00	0.00	**-**	0.00	0.00	0.10	-
Accomodation	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Inhibition Induced Spiking	0.00	0.00	2.00	15.00	0.00	0.00	7.50	3.00
Inhibition Induced Bursting	0.00	0.00	78.00	78.00	0.00	0.00	40.00	54.00
Mean	1.55	0.00	33.36	34.59	12.88	16.58	42.00	21.15

Table 5: Absolute accumulated spike timing error over 1000ms for each firing type using both standard and rescaled equations, comparing 32-bit and 16-bit arithmetics with the FP64 reference.

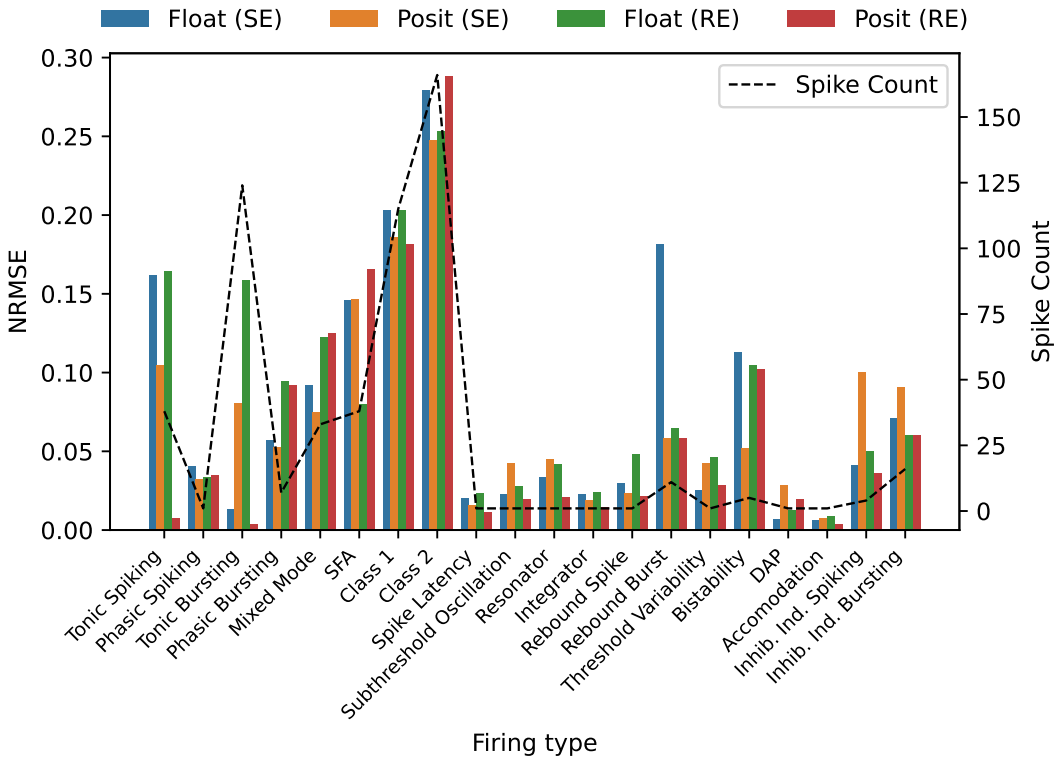


Figure 4: 16-bit floating-point and posit arithmetics using both Standard Equations (SE) and Rescaled Equations (RE). Where Inhib. Ind. Spiking is Inhibition Induced Spiking and Inhib. Ind. Bursting is Inhibition Induced Bursting. Spike count for each simulation is overlaid with its axis on the right-hand side.

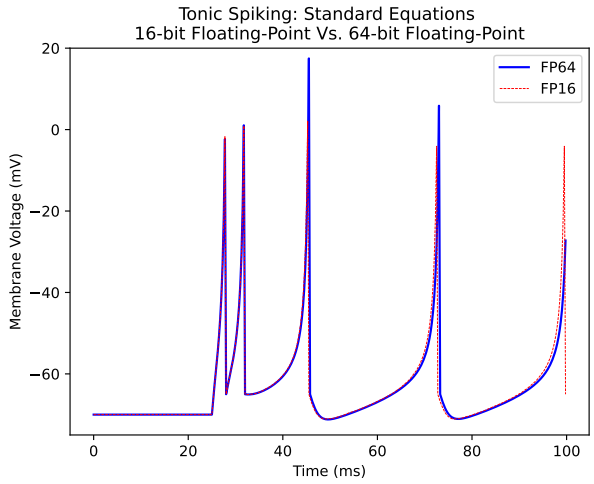


Figure 5

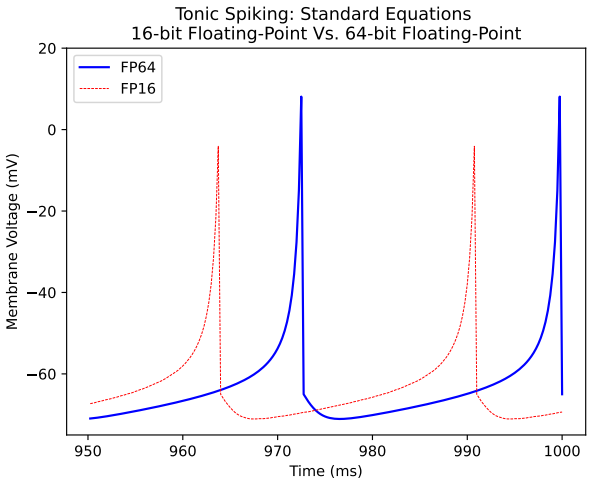


Figure 6

Figure 7: A comparison of FP16 and FP64 arithmetic simulating tonic spiking over 1000ms. (a) shows the first 100ms of the simulation and (b) the final 50ms. Initially in (a), it can be seen that there is good alignment between the FP16 with the FP64 reference. However, as the simulation proceeds, the lag – caused by accumulated arithmetic error – becomes clearly visible in (b).

ple to implement in hardware, it inevitably increases the complexity of the design.

Similarly, Jin et al. [29] also achieved a spiking Izhikevich neuron using only 16-bits on the SpiNNaker system. However, in their work they needed to use two scaling factors and to rewrite the equations (Equation 9 and Equation 10) to better fit with the ARM instruction set. Having gone beyond simply rescaling, it becomes difficult to

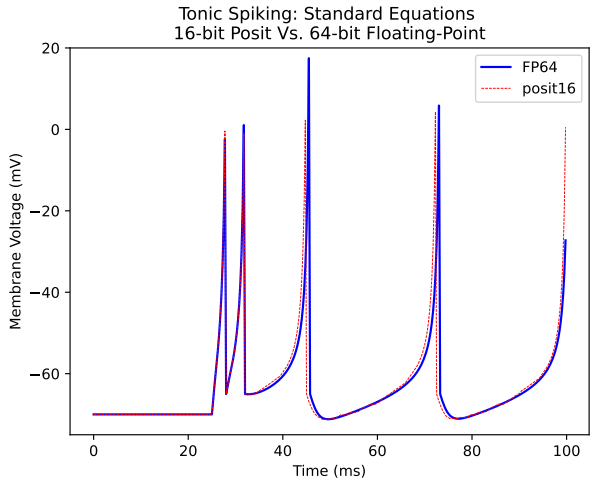


Figure 8

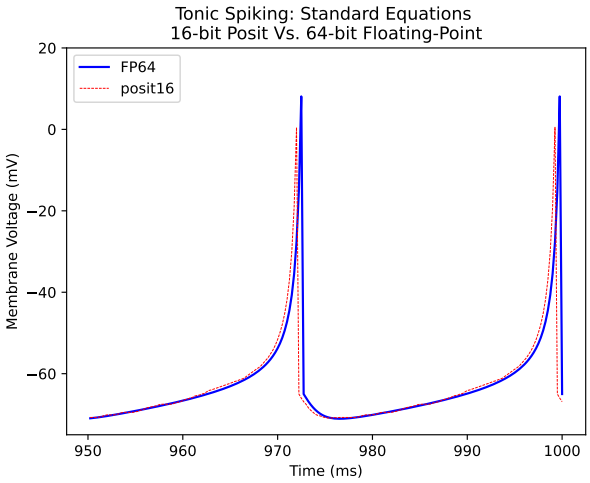
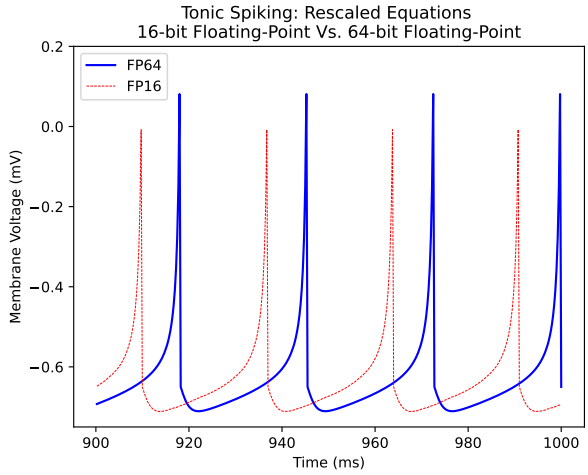
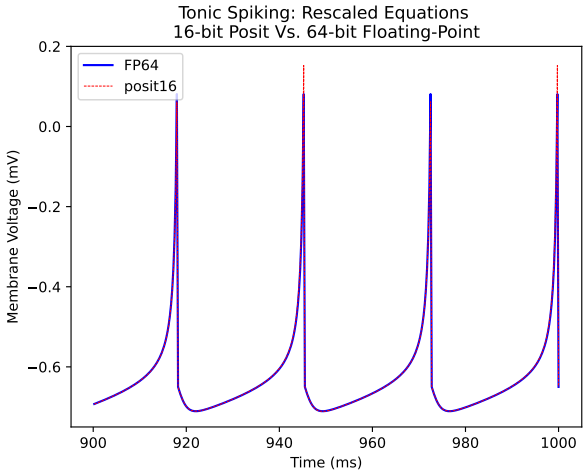


Figure 9

Figure 10: A comparison of posit16 and FP64 arithmetic simulating tonic spiking over 1000ms. (a) shows the first 100ms of the simulation and (b) the final 50ms. Very little spike timing error can be seen in either (a) or (b) meaning posit16 arithmetic has introduced very few errors during the simulation.



(a)



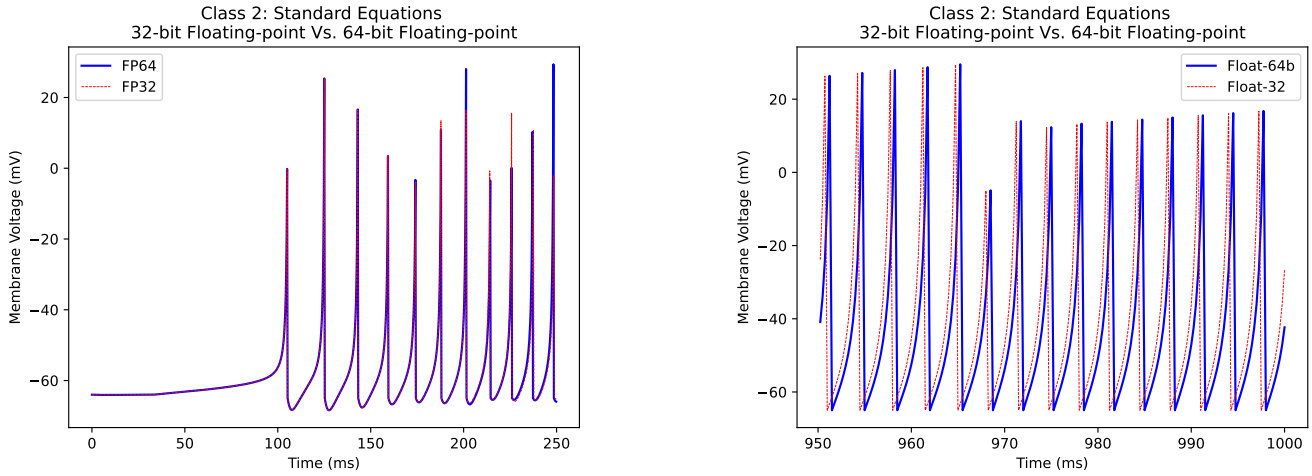
(b)

Figure 11: Both (a) and (b) show the last 50ms of simulation, where (a) employs FP16 and (b) uses posit16 arithmetic when using the rescaled equations. (a) shows that rescaling does not improve the accuracy in the FP16 case. However, (b) demonstrates the elimination of timing errors in posit16 when using rescaled equations.

make comparisons with other Izhikevich implementations. Moreover, they did not assess their results for accuracy, making their work hard to compare against and difficult to replicate on non-ARM hardware. Nor did they attempt to reproduce all 20 firing patterns. Hence, while it has been previously demonstrated that 16-bit Izhikevich models are *possible* using 16-bit fixed-point arithmetic, it has not been shown that this is possible without either changing the equations considerably, or using a much more complicated system to ameliorate the arithmetic errors. Nor has it been shown that at reduced precision, all firing types are possible. Hence, we have not considered fixed-point in this work.

4. Conclusion

The objective of this study was to examine the feasibility of employing posit arithmetic as an alternative to FP in running SNN simulations using Izhikevich neurons. Additionally, the study aimed to explore the effect on accuracy of reducing the bit depth in both number systems. Typically, reducing the bit-depth has several advantages such as speed and power efficiency, but is often deleterious to accuracy. Hence, the impact of rescaling the equations, as a potential approach to mitigating this reduced accuracy was also investigated. This is the first study to establish quantitatively how posit arithmetic differs from FP in this context. Our research shows that there is very little dif-



(a) (b)

Figure 12: Class 2 using FP32 and standard equations. (a) shows the first 250ms of the simulation and (b) the final 50ms. A slight timing lag can be seen in (b).

ference between number systems at 32-bit either in terms of membrane voltage or spike timing. However, when the bit-depth was reduced to 16-bit errors become detectable, but generally posit arithmetic is more accurate than FP16. Most notably, we showed the advantage of rescaling the Izhikevich equations when using posit arithmetic, especially for tonic firing. These results imply that any future SNN hardware development may want to overlook a dedicated floating-point unit and first consider a posit based system. This could allow for more efficient utilisation of silicon resources in such systems.

While this study provides several important positive contributions, it suffers from a number of limitations. Notably, only one neuron type was tested. Reducing or increasing the complexity of the equations will likely have an impact on the performance of both arithmetic systems. A further limitation is the use of a single ODE solver and keeping the update step size to the original values defined by Izhikevich. Both of these variables have previously been show to have a significant impact on SNN accuracy [23]. Additionally, this study only explored one mitigation technique – rescaling by a constant factor. It might be expected that different scaling factors could affect accuracy to differing extents, as was found in Klöwer et al. [30]. Other mitigation techniques are also possible but were not explored here, such as using mixed precision with a larger bit-depth for variables which require a higher precision or intermediate results which can be outside of the optimal range for a particular arithmetic. This might be addressed by reordering or precomputing various values, and this was not considered in this work. Nevertheless, this study offers some important contributions and highlights posit arithmetic as an interesting avenue of future SNN research. We therefore suggest future work attempt to establish the importance of ODE solver choice and time-step size on accuracy while running posit-based SNN simulations.

Acknowledgements

TFH was part funded by Sundance Multiprocesssor Ltd., UK. and an EPSRC Doctoral Training Partnerships (DTP) grant. JK was funded by the EPSRC (grant EP/V052241/1)

References

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, July 2019. doi: 10.1109/IEEESTD.2019.8766229. Conference Name: IEEE Std 754-2019 (Revision of IEEE 754-2008).

[2] W. C. Abraham, O. D. Jones, and D. L. Glanzman. Is plasticity of synapses the mechanism of long-term memory storage? *npj Sci. Learn.*, 4(1):9, Dec. 2019. ISSN 2056-7936. doi: 10.1038/s41539-019-0048-y.

[3] K. Akbarzadeh-Sherbaf, B. Abdoli, S. Safari, and A.-H. Vahabie. A Scalable FPGA Architecture for Randomly Connected Networks of Hodgkin-Huxley Neurons. *Front Neurosci*, 12:698, Oct. 2018. ISSN 1662-4548. doi: 10.3389/fnins.2018.00698.

[4] A. S. Alkabaa, O. Taylan, M. T. Yilmaz, E. Nazemi, and E. M. Kalmoun. An Investigation on Spiking Neural Networks Based on the Izhikevich Neuronal Model: Spiking Processing and Hardware Approach. *Mathematics*, 10(4):612, 2022. doi: https://doi.org/10.3390/math10040612.

[5] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll. A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks. *Front. Neurorobot.*, 12:35, July 2018. ISSN 1662-5218. doi: 10.3389/fnbot.2018.00035.

[6] Y. Cao, Y. Chen, and D. Khosla. Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition. *Int J Comput Vis*, 113(1):54–66, May 2015. ISSN 1573-1405. doi: 10.1007/s11263-014-0788-3. URL `https://doi.org/10.1007/s11263-014-0788-3`.

[7] R. Chaurasiya, J. Gustafson, R. Shrestha, J. Neudorfer, S. Nambiar, K. Niyogi, F. Merchant, and R. Leupers. Parameterized Posit Arithmetic Hardware Generator. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 334–341. IEEE, Oct. 2018. ISBN 978-1-5386-8477-1. doi: 10.1109/ICCD.2018.00057. event-place: Orlando, FL, USA.

[8] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1):82–99, Jan. 2018. ISSN 1937-4143. doi: 10.1109/MM.2018.112130359.

[9] A. Dawson and P. D. Düben. rpe v5: an emulator for reduced floating-point precision in large numerical simulations. *Geoscientific Model Development*, 10(6):2221–2230, June 2017. ISSN 1991-959X. doi: 10.5194/gmd-10-2221-2017. Publisher: Copernicus GmbH.

[10] L. Forget and Y. Uguen. Comparing posit and IEEE-754 hardware cost. page 13.

[11] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol. A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28nm CMOS. *IEEE Trans. Biomed. Circuits Syst.*, pages 1–1, 2018. ISSN 1932-4545, 1940-9990. doi: 10.1109/TBCAS.2018.2880425. arXiv: 1804.07858 [cs].

[12] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The SpiNNaker Project. *Proceedings of the IEEE*, 102(5):652–665, May 2014. ISSN 1558-2256. doi: 10.1109/JPROC.2014.2304638.

[13] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama. Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems. *Frontiers in Neuroscience*, Mar. 2021. ISSN 16624548. doi: http://dx.doi.org/10.3389/fnins.2021.638474.

[14] J. Gustafson, G. Bohlender, S. Y. Chung, and V. Dimitrov. Standard for Posit™ Arithmetic (2022), Mar. 2022.

[15] J. L. Gustafson and I. T. Yonemoto. Beating Floating Point at its Own Game: Posit Arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, Apr. 2017. ISSN 2313-8734. doi: 10.14529/jsfi170206.

[16] J. Hauser. Berkeley SoftFloat, 2018. URL `http://www.jhauser.us/arithmetic/SoftFloat.html`.

[17] M. Hayati, M. Nouri, S. Haghiri, and D. Abbott. Digital Multiplierless Realization of Two Coupled Biological Morris-Lecar Neuron Model. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(7):1805–1814, July 2015. ISSN 1558-0806. doi: 10.1109/TCSI.2015.2423794. Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers.

[18] M. Heidarpour, A. Ahmadi, and R. Rashidzadeh. A CORDIC Based Digital Hardware For Adaptive Exponential Integrate and Fire Neuron. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(11):1986–1996, Nov. 2016. ISSN 1558-0806. doi: 10.1109/TCSI.2016.2598161.

[19] M. Heidarpur, A. Ahmadi, and N. Kandalaft. A digital implementation of 2D Hindmarsh–Rose neuron. *Nonlinear Dynamics*, 89(3):2259–2272, Aug. 2017. ISSN 1573-269X. doi: 10.1007/s11071-017-3584-0.

[20] M. Heidarpur, A. Ahmadi, and M. Ahmadi. Time Step Impact on Performance and Accuracy of Izhikevich Neuron: Software Simulation and Hardware Implementation. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Oct. 2020. doi: 10.1109/ISCAS45731.2020.9180632.

[21] M. Heidarpur, S. Member, P. Khosravifar, A. Ahmadi, and S. Member. CORDIC-Astrocyte: Tripartite Glutamate-IP3-Ca2+ Interaction Dynamics on FPGA. *IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS*, 14 (1):36–47, 2020.

[22] L. Hodgkin and A. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J.Physiol*, 117(4):500–544, Aug. 1952.

[23] M. Hopkins and S. Furber. Accuracy and Efficiency in Fixed-Point Neural ODE Solvers. *Neural Computation*, 27(10):2148–2182, Oct. 2015. ISSN 0899-7667.

[24] M. Hopkins, M. Mikaitis, D. R. Lester, and S. Furber. Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190052, Mar. 2020. ISSN 1364-503X, 1471-2962. doi: 10.1098/rsta.2019.0052.

[25] E. Izhikevich. Simple model of spiking neurons. *IEEE Trans. Neural Netw.*, 14(6):1569–1572, Nov. 2003. ISSN 1045-9227. doi: 10.1109/TNN.2003.820440.

[26] E. Izhikevich. Figure1.m, 2004. URL `http://www.izhikevich.org/publications/figure1.m`.

[27] E. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, Sept. 2004. ISSN 1941-0093. doi: 10.1109/TNN.2004.832719. Conference Name: IEEE Transactions on Neural Networks.

[28] M. K. Jaiswal and H. K.-H. So. PACoGen: A Hardware Posit Arithmetic Core Generator. *IEEE Access*, 7:74586–74601, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2920936.

[29] X. Jin, S. B. Furber, and J. V. Woods. Efficient modelling of spiking neural networks on a scalable chip multiprocessor. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2812–2819, June 2008. doi: 10.1109/IJCNN.2008.4634194. ISSN: 2161-4407.

[30] M. Klöwer, P. D. Düben, and T. N. Palmer. Number Formats, Error Mitigation, and Scope for 16-Bit Arithmetics in Weather and Climate Modeling Analyzed With a Shallow Water Model. *Journal of Advances in Modeling Earth Systems*, 12(10):e2020MS002246, 2020. ISSN 1942-2466. doi: 10.1029/2020MS002246.

[31] J. C. Knight and T. Nowotny. Larger GPU-accelerated brain simulations with procedural connectivity, Apr. 2020. Pages: 2020.04.27.063693 Section: New Results.

[32] V. Leon, T. Paparouni, E. Petrongonas, D. Soudris, and K. Pekmestzi. Improving Power of DSP and CNN Hardware Accelerators Using Approximate Floating-point Multipliers. *ACM Transactions on Embedded Computing Systems*, 20(5):39:1–39:21, July 2021. ISSN 1539-9087. doi: 10.1145/3448980.

[33] C. Leong. Files · master · Cerlane Leong / SoftFloat-Python · GitLab, Feb. 2019. URL

https://gitlab.com/cerlane/SoftFloat-Python/-/tree/master.
[34] C. Leong. Cerlane Leong / SoftPosit · GitLab, 2022. URL https://gitlab.com/cerlane/SoftPosit.
[35] D. Mallasén, R. Murillo, A. A. D. Barrio, G. Botella, L. Piñuel, and M. Prieto-Matias. PERCIVAL: Open-Source Posit RISC-V Core With Quire Capability. *IEEE Transactions on Emerging Topics in Computing*, 10(3):1241–1252, July 2022. ISSN 2168-6750. doi: 10.1109/TETC.2022.3187199. Conference Name: IEEE Transactions on Emerging Topics in Computing.
[36] S. M. Mishra, A. Tiwari, H. S. Shekhawat, P. Guha, G. Trivedi, P. Jan, and Z. Nemec. Comparison of Floating-point Representations for the Efficient Implementation of Machine Learning Algorithms. In *2022 32nd International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 1–6, Apr. 2022. doi: 10.1109/RADIOELEKTRONIKA54537.2022.9764927.
[37] S. W. Moore, P. J. Fox, S. J. Marsh, A. T. Markettos, and A. Mujumdar. Bluehive - A field-programable custom computing machine for extreme-scale real-time neural network simulation. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 133–140, Apr. 2012. doi: 10.1109/FCCM.2012.32.
[38] R. Omidi and S. Sharifzadeh. Design of low power approximate floating-point adders. *International Journal of Circuit Theory and Applications*, 49(1):185–195, 2021. ISSN 1097-007X. doi: 10.1002/cta.2831. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.2831. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cta.2831.
[39] D. E. Oorschot. Total number of neurons in the neostriatal, pallidal, subthalamic, and substantia nigral nuclei of the rat basal ganglia: a stereological study using the cavalieri and optical disector methods. *The Journal of Comparative Neurology*, 366(4): 580–599, Mar. 1996. ISSN 0021-9967. doi: 10.1002/(SICI)1096-9861(19960318)366:4¡580::AID-CNE3¿3.0.CO;2-0.
[40] Q. T. Pham, T. Q. Nguyen, P. C. Hoang, Q. H. Dang, D. M. Nguyen, and H. H. Nguyen. A review of SNN implementation on FPGA. In *2021 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, pages 1–6, Oct. 2021. doi: 10.1109/MAPR53640.2021.9585245.
[41] S. Rezaei, R. Omidi, and A. Azarpeyvand. Logarithm-approximate floating-point multiplier. *Microelectronics Journal*, 127:105521, Sept. 2022. ISSN 0026-2692. doi: 10.1016/j.mejo.2022.105521.
[42] B. Sen-Bhattacharya, S. James, O. Rhodes, I. Sugiarto, A. Rowley, A. B. Stokes, K. Gurney, and S. B. Furber. Building a Spiking Neural Network Model of the Basal Ganglia on SpiNNaker. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):823–836, Sept. 2018. ISSN 2379-8939. doi: 10.1109/TCDS.2018.2797426. Conference Name: IEEE Transactions on Cognitive and Developmental Systems.
[43] V. H. L. Silva, J. F. Chaves, R. M. Gomes, and B. A. Santos. Posit-based Spiking Neuron in an FPGA. page 4, 2021.
[44] M. J. Skocik and L. N. Long. On the Capabilities and Computational Costs of Neuron Models. *IEEE Trans. Neural Netw. Learning Syst.*, 25(8):1474–1483, Aug. 2014. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2013.2294016.
[45] A. Tamura, T. Ueta, and S. Tsuji. Bifurcation analysis of Izhikevich model.
[46] T. P. Vogels and L. F. Abbott. Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons. *J. Neurosci.*, 25(46):10786–10795, Nov. 2005. ISSN 0270-6474, 1529-2401. doi: 10.1523/JNEUROSCI.3508-05.2005.
[47] R. Wang and A. van Schaik. Breaking Liebig's Law: An Advanced Multipurpose Neuromorphic Engine. *Frontiers in Neuroscience*, 12, 2018. ISSN 1662-453X.
[48] R.-J. Zhu, Q. Zhao, and J. K. Eshraghian. SpikeGPT: Generative Pre-trained Language Model with Spiking Neural Networks, Feb. 2023. arXiv:2302.13939 [cs].