Spiking P System simulator for image recognition

1 Rules codification and application

How the simulator executes and encodes the rules is of fundamental importance in the project. Each neuron in the CSV has a list of values called id, $initialCharge$, $outputTargets$, $neuronType$, $rules$, and the rules are theoretically unlimited in number, defined as tuples of the form $[div, mod, source, target, delay]$. Let's examine the elements.

- **Regex definition** div and mod make it possible to describe the regular expression present in the formula, which must match the number of spikes available. These spikes form an alphabet consisting solely of a's (for example, a neuron with 6 spikes has the alphabet aaaaaa). A rule in such an alphabet can always be written as $a^{mod}(a^{div})^*$, where the exponents are the values from the tuple used to describe it. For example, the regex $a(aa)^+a$ can be rewritten as $aaaa(aa)^*$ both through mathematical transformation and because both expressions represent the same concept: accept all even numbers greater than 2.

- **Rule application** To be described, a rule also needs a value that defines the number of spikes that will be consumed upon its application, and thus required for firing. To apply a rule, you need the values $mod, div, source$ of the tuple $[div, mod, source, target, delay]$, where $charge$ is the internal charge of the neuron, which must match the regex. The code to evaluate the regex is:

```
def check(self, charge):
    if charge > 0 and charge >= self.mod:
        if self.div > 0:
            return charge >= self.source and (charge - self.
                mod) % self.div == 0
        if self.div == 0:
            return charge >= self.source and charge == self.
                mod
    return False
```

Listing 1: `check` Method

The charge is checked to ensure it is not negative, because a neuron with no spikes will never fire. It is also checked whether it is greater than mod, because no formula of the type $a^{mod}(a^{div})^*$ can be matched by an expression of the form a^{charge} if $charge < mod$. With this formula, it is therefore possible to apply all such regexes.

Note to keep in mind: for now, if a rule contains 0 in the source value, this is interpreted as 'consume all spikes, therefore set to 0'. This was implemented

as a practical necessity during the first firing phase of rules and would require an extension of the formalism allowing regular expressions over consumed charges, rather than just fixed values. Thus, instead of multiplying the rules $16 - x$ times like $(a^{>=14})/a^{14} - > a; 0; (a^{>=15})/a^{15} - > a; 0$, we use $(a^{>=13})/a^{*} - > a; 0$, also because the former form leaves residual charges in the neuron, which alter the subsequent image. This problem arises only in a fast network analyzing one image per step, which is currently the test model, but may be changed. To solve this issue, it is sufficient to define a forgetting function that accepts values below the limit and deletes all present spikes.

- **Target** The fourth value, *target*, can be 0 or 1. 0 indicates a forgetting rule, while 1 indicates a firing rule.
- **Delay** The last value of the tuple is the number of steps required for the neuron to activate the rule, during which the neuron remains inactive.

2 Energy costs

To obtain a meaningful estimate of the network's energy cost, the type and number of operations required to process an image are calculated. Unlike ANNs and SNNs, which typically perform operations in floating point, this model uses only boolean values (at the synaptic level) and integers (for spikes within neurons). This significantly reduces computational costs. However, one must consider the energy cost of evaluating the regular expression associated with each neuron—a cost not present in the two models mentioned above.

Operation	**Energy cost**	**Reference**
Switching	Min: 0.0021 fJ Max: 0.047 fJ	See Fig.1 in [1]
A single logic gate (AND, OR, NOT, ...) on 1 bit	From 0.1 fJ to 1 fJ	in [2], that cite [3] and [4].
Simple regular expressions (==, less then, %2)	From 1 fJ to 10 fJ	again in [4]
INT4	Min: 250 fJ Max: 1.6 pJ	See Fig.1 in [1]
FP32	Min: 4.8 pJ Max: 63 pJ OR from 0.9 pJ to 3.7 pJ	See Fig.1 in [1] OR table IV in [5]

Table 1: Joule costs of some operations in CMOS logic

It is worth mentioning that SNNs can obtain much higher energy-efficiency on neuromorphic hardware platform

Model Component	**Estimated Cost**	**Operation**	**Upper bound**
Boolean spike sum	∼0.1–1 fJ	A single logic gate for each incoming spike	n° synapsis
Simple neuron REGEX (e.g. $\geq N$ or ==N)	∼1–10 fJ	Simple regular expression	all complex neurons
Complex neuron REGEX (e.g., "multiple of 4 but not 6")	∼10–100 fJ	Small logic network: combination of multiple gates and conditions	n° neurons * n° rules containing

Table 2: Energy estimates for basic computational operations in the model, based on typical CMOS logic gate consumption.

Worst-case estimate

- Each synapse fires at every step, consuming approximately 1 fJ per spike.
- Each neuron evaluates complex regular expressions at every step, with an estimated cost of 100 fJ per rule.

- Assuming the entire model contains nr rules to be checked at each step, and there are three channels for each input images:

$$E_{\text{worst}} = 3 \cdot (N_{\text{synapses}} \cdot 1\,\text{fJ} + 100\,\text{fJ} \cdot nr)$$

Case-specific estimate (realistic scenario)

- If the dataset contains mostly 1s, after training, the synaptic matrix and rule thresholds can be inverted, allowing inversion of the input data as well. This inversion ensures that no more than 50% of synapses are activated during input image processing. In deeper layers, this assumption may not hold, but rule thresholds are calibrated to maintain appropriate firing rates. Thus, assuming that roughly 50% of synapses are active, and each active synapse costs 0.5 fJ:

$$E_{\text{synapses}} = 0.5 \cdot N_{\text{synapses}} \cdot 0.5\,\text{fJ}$$

- The rules in the current P system are simple threshold checks, costing about 10 fJ each. However, as noted in [6]: "In SNPSs, neurons only become active when they accumulate enough spikes to trigger a rule, which allows many neurons to remain inactive, further reducing energy consumption. This sparse and asynchronous activity is a defining characteristic that enhances the efficiency of SNPSs.". Therefore, only neurons that actually fire contribute to the energy cost. Assuming about 50% of neurons are active at each step:

$$E_{\text{neurons}} = N_{\text{neurons}} \cdot 0.5 \cdot 10\,\text{fJ}$$

$$E_{\text{total}} = 3 \cdot (0.5 \cdot N_{\text{synapses}} \cdot 0.5\,\text{fJ} + 0.5 \cdot N_{\text{neurons}} \cdot 10\,\text{fJ})$$

- Which, in our specific case, turns out to be:

$$E_{\text{total}} = 3 \cdot (0.5 \cdot 1176 \cdot 0.5\,\text{fJ} + 0.5 \cdot 841 \cdot 10\,\text{fJ}) = 4500\text{fJ}$$

  Note that, in the code, we have the exact number of spikes generated and rules applied, and we can distinguish between firing rules and forgetting rules, so the value in joules obtained from the code will be more accurate.

ANN comparison

- By examining the work in [5], particularly Table IV, we observe that a 32-bit multiply-and-accumulate (MAC) operation consumes approximately 4.6 pJ of energy. Based on this, we can estimate the energy required by the model if it were implemented as a fully connected network:

$$E_{\text{ANN}} = (N_{\text{layer1}} \cdot N_{\text{layer2}} + N_{\text{layer2}} \cdot N_{\text{classes}}) \cdot 4.6\,\text{pJ}$$

- However, our current code instead uses a convolutional layer as the first layer, followed by a fully connected (FC) layer as the second. Therefore, the energy cost becomes:

$$E_{\text{CNN+FC}} = (N_{\text{layer1}} + N_{\text{layer2}} \cdot N_{\text{classes}}) \cdot 4.6\,\text{pJ}$$

- In the specific case considered, a 4×4 convolutional window is applied to a 28×28 input, using a stride of 4, resulting in a 7×7 feature map (i.e., 49 neurons). These are then connected to 8 output classes. This leads to an estimated energy cost of:

$$E_{\text{CNN+FC}} = (28 \cdot 28 + 7 \cdot 7 \cdot 8) \cdot 4.6\,\text{pJ} = 5400\text{pJ}$$

Collected data

Worst energy spent: 813663 pJ.

Expected energy spent: 14414 pJ.

This is for a SNPS with 1000 images as a training set and 1000 for a test set, including the 2 training phases and the classification phase.

References

[1] Sadasivan Shankar and Albert Reuther. "Trends in energy estimates for computing in ai/machine learning accelerators, supercomputers, and compute-intensive applications". In: *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2022, pp. 1–8.

[2] Reza Maram, James van Howe, Deming Kong, Francesco Da Ros, Pengyu Guan, Michael Galili, Roberto Morandotti, Leif Katsuo Oxenløwe, and José Azaña. "Frequency-domain ultrafast passive logic: NOT and XNOR gates". In: *Nature Communications* 11.1 (2020), p. 5839.

[3] Rodney S Tucker and Kerry Hinton. "Energy consumption and energy density in optical and electronic signal processing". In: *IEEE Photonics Journal* 3.5 (2011), pp. 821–833.

[4] David AB Miller. "Attojoule optoelectronics for low-energy information processing and communications". In: *Journal of Lightwave Technology* 35.3 (2017), pp. 346–396.

[5] Youngeun Kim, Joshua Chough, and Priyadarshini Panda. "Beyond classification: Directly training spiking neural networks for semantic segmentation". In: *Neuromorphic Computing and Engineering* 2.4 (2022), p. 044015.

[6] Claudio Zandron. "An Overview on Applications of Spiking Neural Networks and Spiking Neural P Systems". In: *Languages of Cooperation and Communication: Essays Dedicated to Erzsébet Csuhaj-Varjú to Celebrate Her Scientific Career* (2025), pp. 267–278.