Supplementary Information for: DelRec: learning delays in recurrent spiking neural networks

Alexandre Queant[1, 2, 3*], Ulysse Rançon[1, 2], Benoit R Cottereau[1, 2, 3†], Timothée Masquelier[1, 2†]

[1]CerCo, CNRS UMR 5549, Toulouse, France.
[2]Université de Toulouse, France.
[3]IPAL, CNRS IRL 2955, Singapore.

*Corresponding author(s). E-mail(s): alexandre.queant@cnrs.fr;
Contributing authors: ulysse.rancon@cnrs.fr; benoit.cottereau@cnrs.fr;
timothee.masquelier@cnrs.fr;
†These authors contributed equally to this work.

S1 Hyperparameters

We detail here all the hyperparameters that we used to perform the experiments presented in this work.

S2 The case of the SHD dataset

S2.1 SHD dataset

The SHD (Spiking Heidelberg Digits) dataset [1] is currently one of the most, if not the most, widely used dataset in the SNN community to benchmark spatiotemporal processing. It is a much smaller dataset than SSC or PS-MNIST, with 10k recordings of spoken digits ranging from zero to nine, in English and German. While the SHD dataset has historically been a valuable tool to help SNNs evolve towards better classification performances, we argue that this dataset has reached saturation, and became obsolete in terms of benchmarking, for the two following reasons:

- SHD lacks a dedicated validation set, and historical evaluations of SNN performance on this dataset have relied solely on the test set. This approach is methodologically flawed and leads to an overfitting of the test set. More recent works have set more

Table S1: General hyperparameters used for the SSC and PS-MNIST datasets.

Hyperparameter	**SSC**	**PS-MNIST**
`epochs`	100	200
`batch size`	128	256
`learning rate max weights`	1e-3	1e-3
`learning rate max positions`	5e-2	5e-2
`optimizer`	Adam	AdamW
`weight decay`	1e-5	1e-2
`batch norm`	`False`	`False`
`bias`	`False`	`False`
`feedforward dropout`	0.1	0.1
`recurrent dropout`	0.3	0.2
`scheduler weights`	One cycle	One cycle
`scheduler positions`	One cycle	One cycle

Table S2: Neuron-related hyperparameters used for the SSC and PS-MNIST datasets.

Hyperparameter	**SSC**	**PS-MNIST**
τ (time-steps)	2	2
$V_{threshold}$	1.	1.
`reset type`	Soft	Soft
V_{reset}	None	None
`detach reset`	`False`	`False`
surrogate function	Triangle	Triangle

rigorous standards by using a fraction of the training set as a validation set, before reporting the best model's accuracy on the test set [2, 3].

- While the best models report around 93% of accuracy on the test set (using a clean split), [3] explain that further improvements in performance are likely not statistically significant given the small size of the test set (2264): with naive assumptions on error rates, the Bayesian confidence intervals of accuracies over 93% overlap.

For all these reasons, we decided not to include SHD in Table **??**, and we recommend its use only as an initial validation step for proof-of-concept studies. Nevertheless, we evaluated DelRec on the SHD dataset, and we provide in this Appendix the results and the methodology these tests.

S2.2 Methodology

For this dataset, we also reduced the input size by binning every 5 neurons of the original 700, resulting in a spatial input dimension of 140. As well, we binned the inputs temporally, using a discrete time-step of $\Delta = 10$ms. The architecture used is composed of two fully connected hidden layers, and each of these layers can be followed by batch normalization, a neuron module, and dropout. The readout consists of $n_{\text{classes}} = 20$ LIF neurons with an infinite threshold. Following [4], we consider V_i^r the membrane potential of the readout neuron i, then we compute the softmax over output neurons at each time-step before summing across the temporal dimension. Accordingly, the model's output for this neuron is:

$$\hat{y}_i = \sum_{t \in T} \text{softmax}(V_i^r[t]) = \sum_{t \in T} \frac{e^{V_i^r[t]}}{\sum_{j=1}^{n_{\text{classes}}} e^{V_j^r[t]}} \tag{S1}$$

with T the temporal dimension. Then, denoting N the batch size and y the ground truth, we compute the cross-entropy loss for one batch as:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} -\log(\hat{y}_{n,\ y_n}[n]). \tag{S2}$$

We used augmentations on the training set of the SHD in order to limit overfitting. We use this method only on our large models when trying to reach high accuracies, not in our ablation study. We followed the approach of [5] and [3]:

- We applied a random temporal shifting to inputs, with a shift uniformly drawn in the interval $\big[-100 \text{ time steps},\ 100 \text{ time steps}\big]$.
- Each input was blended with an input from the same class by aligning their center of mass and, for each time step, picking a spike from either sample with a probability of 0.5.

We initialized the feedforward weights using Kaiming uniform initialization, with $a = \sqrt{5}$:

$$w_{ij} \sim \mathcal{U}\left(-\sqrt{\tfrac{6}{\text{in\_features}}},\ +\sqrt{\tfrac{6}{\text{in\_features}}}\right),$$
$$b_i \sim \mathcal{U}\left(-\tfrac{1}{\sqrt{\text{in\_features}}},\ +\tfrac{1}{\sqrt{\text{in\_features}}}\right),$$

where in_features denotes the number of input units to the layer. The recurrent delays were initialized with a random draw from a uniform distribution:

$$d_i^{\text{rec}} \sim \mathcal{U}(10,\ 30).$$

The hyperparameters used are presented in Table S3 and Table S4.

S2.3 Results

We present the performance of DelRec on the SHD dataset in Table S5. The model achieves state-of-the-art performance, with a significantly low number of parameters.

Table S3: General hyperparameters used for the SHD dataset.

Hyperparameter	**SHD**
`epochs`	150
`batch size`	256
`learning rate max weights`	5e-3
`learning rate max positions`	5e-2
`optimizer`	AdamW
`weight decay`	1e-5
`batch norm`	`True`
`bias`	`True`
`feedforward dropout`	0.4
`recurrent dropout`	0.2
`scheduler weights`	One cycle
`scheduler positions`	Cosine annealing

Table S4: Neuron-related hyperparameters used for the SHD dataset.

Hyperparameter	**SHD**
τ (time-steps)	1.005
$V_{threshold}$	1.
`reset type`	Hard
V_{reset}	0
`detach reset`	`True`
surrogate function	Arctan

S3 Recurrent delays mitigate the risk of exploding gradients

The purpose of this section is to provide a more detailed explanation, without resorting to full mathematical formalism, of why introducing delays in recurrent connections reduces the risk of exploding gradients in these connections.

Let's consider the generic equation of a neuron with a simple recurrent connection to itself, in the case of a soft reset. We allow the recurrent connection to be delayed by d time-steps, with $d \geq 0$:

$$H[t+1] = \alpha H[t] + I[t+1] + W_{rec} S[t-d] - \theta S[t] \tag{S3}$$

Table S5: **Classification accuracies on SHD using a clean split**, ranked by accuracy.

Model	Rec.	Rec. Delays	Ff. Delays	LIF	Param	Test Acc. [%]
BRF [6]	✓				0.1M	92.70 ± 0.70
SE-adLIF (1L) [2]	✓				37.5k	93.18 ± 0.74
EventProp*[b]* [3]	✓	✓	✓	✓	∼1M*[a]*	93.24 ± 1.00
DelRec (Only Rec. delays) ***Ours[b]***	✓	✓		✓	**0.17M**	93.39 ± 0.45
DCLS*[b]* [7]			✓	✓	0.22M	93.77 ± 0.68
SE-adLIF (2L) [2]	✓				0.45M	93.79 ± 0.76

[a] Estimated from Figure 5.
[b] Models trained using augmentations.

with $S[t] = \Theta(H[t] - \theta)$. Denoting $\mathcal{L}$ the loss function, we can write:

$$\frac{\partial \mathcal{L}}{\partial W_{rec}} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}}{\partial H[t]} \cdot \frac{\partial H[t]}{\partial W_{rec}} = \sum_{t=0}^{T} \delta[t] \cdot \epsilon[t]. \tag{S4}$$

with $\delta_t = \frac{\partial \mathcal{L}}{\partial H[t]}$ and $\epsilon_t = \frac{\partial H[t]}{\partial W_{rec}}$.

We focus on the ϵ terms. Using Eq. S3, we derive:

$$\epsilon_{t+1} = \alpha \epsilon_t + S[t-d] + W_{rec} \frac{\partial S[t-d]}{\partial W_{rec}} - \theta \frac{\partial S[t]}{\partial W_{rec}}, \tag{S5}$$

Then, noticing that:

$$\frac{\partial S[t]}{\partial W_{rec}} = \frac{\partial S[t]}{\partial H[t]} \frac{\partial H[t]}{\partial W_{rec}} = \psi'(H[t] - \theta)\epsilon_t \tag{S6}$$

with ψ our surrogate function, the relation can be rewritten:

$$\epsilon_{t+1} = \alpha \epsilon_t + S[t-d] + W_{rec}\psi'(H[t-d] - \theta)\epsilon_{t-d} - \theta\psi'(H[t] - \theta)\epsilon_t. \tag{S7}$$

Using $\Psi'_t = \psi'(H[t] - \theta)$, we find the following recurrence:

$$\epsilon_{t+1} = (\alpha - \theta\Psi'_t)\epsilon_t + W_{rec}\Psi'_{t-d}\epsilon_{t-d} + S[t-d]. \tag{S8}$$

At this stage, we simplify the notations by setting $A_t = (\alpha - \theta\Psi'_t)$ and $B_t = W_{rec}\Psi'_t$, to get the simple recurrent relation:

$$\epsilon_{t+1} = A_t\epsilon_t + B_{t-d}\epsilon_{t-d} + S[t-d]. \tag{S9}$$

One can notice that the range of values taken by A_t and B_t is finite, so we can set:

$$a = \sup_t |A_t|,\ b = \sup_t |B_t| \tag{S10}$$

And have:

$$|\epsilon_{t+1}| \leq a|\epsilon_t| + b|\epsilon_{t-d}|. \tag{S11}$$

We admit here that the asymptotic behavior of $\{\epsilon_t\}$ depends solely on the homogeneous term $A_t\epsilon_t + B_{t-d}\epsilon_{t-d}$, so it suffices to focus on studying the sequence:

$$a\epsilon_t + b\epsilon_{t-d} = \epsilon_{t+1}. \tag{S12}$$

with positive initial conditions in order to get the asymptotic behavior of $\{\epsilon_t\}$. Thus we define the positive sequence $\{e_t\}$ with the same initial conditions as $\{\epsilon_t\}$, ie $e_0 = |\epsilon_0|, ..., e_d = |\epsilon_d|$, and such that:

$$e_{t+1} = ae_t + be_{t-d}. \tag{S13}$$

Its characteristic equation is:

$$\lambda^{t+1} = a\lambda^t + b\lambda^{t-d} \iff \lambda^{d+1} - a\lambda^d - b = 0 \iff \lambda^d(\lambda - a) = b. \tag{S14}$$

Let $\lambda_1, \ldots, \lambda_{d+1}$ be the roots of Eq.S14, counted with multiplicities. We know that, if all roots are simple, the general solution of Eq.S13 is of the form:

$$e_t = \sum_{k=1}^{d+1} c_k \lambda_k^t \tag{S15}$$

and if λ is a root of multiplicity $m > 1$, then its contribution to e_t is:

$$(c_0 + c_1 t + \ldots\ c_{m-1}t^{m-1})\lambda^t. \tag{S16}$$

Therefore, the asymptotic behavior of $\{e\}$ depends on $r_d = \max_j |\lambda_j|$, which is the spectral radius of the characteristic equation.

If all roots satisfy $|\lambda_j| < 1$, then naturally $e_t \longrightarrow 0$. We can also show that it suffices for our gradients ϵ_t to vanish, but we focus here on the gradient explosion. Now, if we are in the case where at least one $|\lambda_j| > 1$, then the sequence $\{e_t\}$ is threatened by explosion. In that case, and admitting that r_d is a real simple root of Eq.S14, we have:

$$r_d^d(r_d - a) = b. \tag{S17}$$

Here, one can notice that as d increases, r_d must decrease for Eq.S17 to hold. In other words, increasing the delay d decreases the spectral radius, which slows down the explosion of the solutions of Eq.S14. By that mean, increasing d also reduces the growth of the sequence $\{e_t\}$ and thus slows down the explosion of ϵ_t.

S4 Computational overhead

The DelRec method enables state-of-the-art performances in dynamic processing with very competitive parameter counts, but it comes at the expense of buffer computations nested in the temporal loop of the recurrent spiking neurons (see Algorithm **??**). While regular Recurrent Spiking Neural Networks layers have a temporal complexity of $O(BS)$ with $S = N_{in}N_{out}$ the number of synapses in the recurrent connections, DelRec increases the complexity to $O(BSL)$ where L is the effective length of the delay buffer (which of the order of $\max_i d_i$ with i the neurons in the layer. The same logic applies for memory overhead. We present in Fig. S1 the peak PGU memory and training time per epoch for the different models, as well as for various delay ranges of the recurrent delay model.

We perform profiling experiments on the SHD dataset (used because of its small size) in order to characterize the computational overhead induced by DelRec. We compare DelRec to a feedforward delay model trained as in [7], a vanilla RSNN and a simple SNN without recurrence. In this experiment, all the networks have 2 hidden layers. For the network using DelRec, only the second hidden layer is recurrent, while for the feedforward delays model, the delays are axonal and only between the first and the second hidden layer. Additional hyperparameters are fixed and shared across models, and are available in the corresponding script of https://github.com/alexmaxad/DelRec.

The curves in Fig. S1a show a significant increase in memory usage when using DelRec as compared to other models, such as the feedforward delay model : for a reasonable number of neurons in the hidden layers (eg. 256), the memory overhead is almost doubled. However, Fig. S1c reveals that for medium sized layers (eg. ≤ 512) the training time for epoch remain fairly close to the ones in the other models without buffer operations, making the method still attractive for medium-sized networks. Yet, for larger layer sizes (above 1024 neurons), the difference in training time per epoch between DelRec and the other models increases exponentially. Additionally, Fig. S1b and Fig. S1d indicate that the memory usage and the training time of the method also heavily depends on the range of the recurrent delays, ie the scheduling buffer size, with memory and temporal complexity increasing exponentially with the buffer length.

These results display the major drawback of our method, namely its memory usage, mainly for the storage of future input values. However, they also demonstrate that for small or medium-sized networks, with a reasonable range of delay values, DelRec's computational overhead remains moderate and comparable to that of other, less performant models. Finally, it is worth mentioning that the DelRec's computational overheads could presumably be much reduced by using low level CUDA kernels instead of the current high level Python code.

In this work, we argue that the additional richness, both in terms of performance and biological modeling, justifies the associated overhead. This is particularly relevant in the context of neuromorphic hardware, where training is performed off-chip and the resulting networks are then deployed on-chip for inference, taking advantage of the chip's programmable delays. Crucially, at inference, the overhead is substantially reduced, collapsing to the intrinsic implementation costs of realizing delays on-chip such as delay-state storage, event buffering, and any associated latency (see [8] for a detailed study of delays implementation costs on-chip).

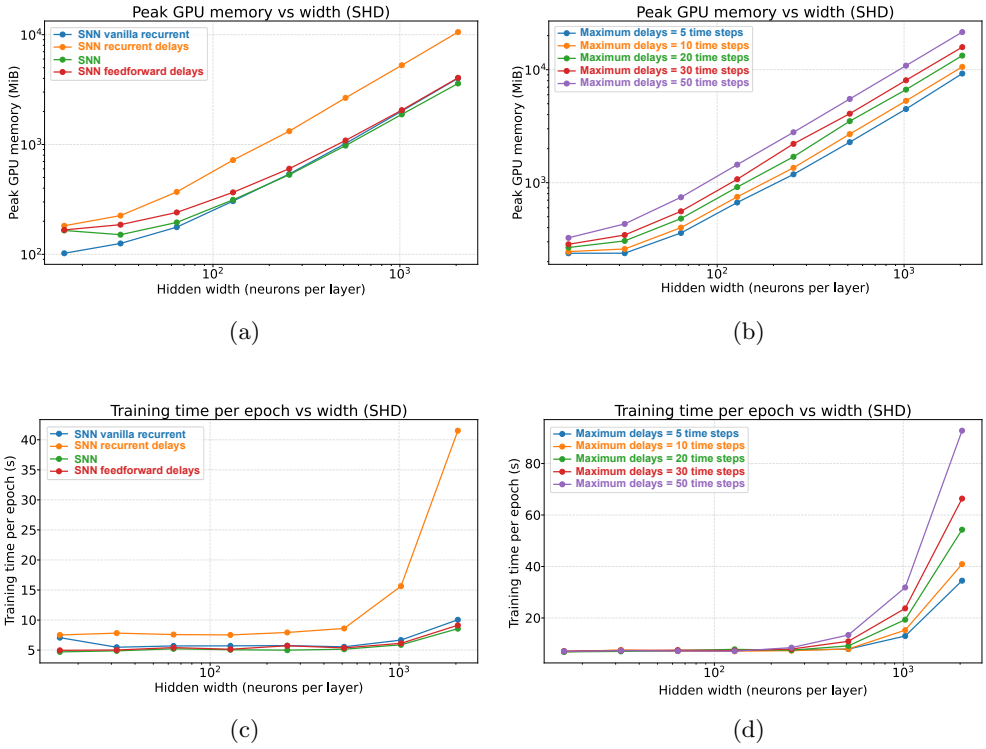

Fig. S1: Computational overhead induced by recurrent delays, compared with other models and with different delay ranges. Recurrent delays models have a fixed $\sigma_{epoch} = 10$ time-steps. For (a) and (c), the recurrent delay model has delays initialized uniformly between 0 and 10. **(a)** Peak GPU memory during one epoch on the SHD dataset, as a function of hidden layer size, for different models. **(b)** Peak GPU memory during one epoch on the SHD dataset, as a function of hidden layer size, for different delay spreads in the recurrent delay model. Each model has recurrent delays distributed uniformly between 0 and "Maximum delays". **(c)** Training time for one epoch on the SHD dataset, as a function of hidden layer sizes, for different models. **(d)** Training time for one epoch on the SHD dataset, as a function of hidden layer sizes, for different delay spreads in the recurrent delay model.

S5 Use of Large Language Models

We declare the following use of LLMs in the preparation of this work:

- For writing: OpenAI's ChatGPT was used in the writing of this article to improve the phrasing and the clarity of some sentences.
- For coding: OpenAI's ChatGPT and GitHub Copilot were used to enhance coding efficiency, more particularly for the factorization of existing scripts and the generation of plots.

References

[1] Cramer, B., Stradmann, Y., Schemmel, J. & Zenke, F. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems* **33**, 2744–2757 (2022).

[2] Baronig, M., Ferrand, R., Sabathiel, S. *et al.* Advancing spatio-temporal processing through adaptation in spiking neural networks. *Nature Communications* **16**, 5776 (2025). URL https://doi.org/10.1038/s41467-025-60878-z.

[3] Mészáros, B., Knight, J. C. & Nowotny, T. Efficient event-based delay learning in spiking neural networks (2025). URL https://arxiv.org/abs/2501.07331. arXiv:2501.07331.

[4] Bittar, A. & Garner, P. N. A surrogate gradient spiking baseline for speech command recognition. *Frontiers in Neuroscience* **16** (2022). URL https://www.frontiersin.org/articles/10.3389/fnins.2022.865897/full.

[5] Nowotny, T., Turner, J. P. & Knight, J. C. Loss shaping enhances exact gradient learning with eventprop in spiking neural networks. *Neuromorphic Computing and Engineering* **5**, 014001 (2025). URL http://dx.doi.org/10.1088/2634-4386/ada852.

[6] Higuchi, S., Kairat, S., Bohte, S. M. & Otte, S. Balanced resonate-and-fire neurons (2024). URL https://arxiv.org/abs/2402.14603. arXiv:2402.14603.

[7] Hammouamri, I., Khalfaoui-Hassani, I. & Masquelier, T. *Learning Delays in Spiking Neural Networks using Dilated Convolutions with Learnable Spacings*, 1–12 (2024). URL http://arxiv.org/abs/2306.17670.

[8] Mészáros, B., Knight, J. C., Timcheck, J. & Nowotny, T. A complete pipeline for deploying snns with synaptic delays on loihi 2 (2025). URL https://arxiv.org/abs/2510.13757. arXiv:2510.13757.