QUADRATIC INTEGRATE-AND-FIRE NEURONS EXHIBIT LESS FRAGMENTED LOSS LANDSCAPES AND OUTPERFORM LEAKY INTEGRATE-AND-FIRE NEURONS IN SPIKE-BASED GRADIENT DESCENT

Anonymous authors
Paper under double-blind review

ABSTRACT

The ability to train spiking neural networks is essential for modeling biological neural networks as well as for neuromorphic computing. However, for the extensively used leaky integrate-and-fire (LIF) neurons, arbitrarily small parameter changes can induce spike (dis)appearances that disrupt subsequent activity, leading to unstable neural representations and permanently silent neurons during exact spike-based gradient descent. Recent work shows that a class of neuron models, which includes the quadratic integrate-and-fire (QIF) neuron, avoids these discontinuities and enables continuous and even smooth spike-based gradient descent. However, it remains unclear whether these advantages translate into practice. Here, we demonstrate that they do so when learning the popular Spiking Heidelberg Digits dataset, via a controlled comparison between networks of LIF and QIF neurons. Specifically, in a first step, we perform a thorough hyperparameter search to optimize both models, revealing a clear performance advantage of QIF neurons. In a second step, we visualize the loss and gradient landscapes and quantify their roughness. Consistent with their inferior performance, we find that the loss landscapes of LIF neurons, which are discontinuous, are rougher and appear more fragmented, and the related gradients are more erratic. An analysis of the landscapes of single samples indicates that these features arise from changes in the temporal order of spikes, which often cause disruptive spike (dis)appearances. Overall, our results advocate, for gradient descent training where precise spike timing matters, to replace LIF neurons with neuron models exhibiting continuous spiking dynamics, such as QIF neurons.

1 INTRODUCTION

Spiking neural networks (SNNs) prevail in biological nervous systems and are essential tools in computational neuroscience (Gerstner et al., 2014; Dayan & Abbott, 2001). Their neurons communicate via brief, stereotypical electrical impulses. This may not be reducible to a communication via spike rates, due to effects that depend on the precise timing of spikes (Gollisch & Meister, 2008; Wolfe et al., 2010; Saal et al., 2016; Sober et al., 2018; Xie et al., 2024). Further, SNNs are used in neuromorphic computing, which promises substantial energy savings, in particular for edge applications (Mehonic & Kenyon, 2022; Kudithipudi et al., 2025). Across these domains, the ability to train SNNs is critical.

The dominant approach for training non-spiking neural networks, hereafter referred to as artificial neural networks (ANNs), is gradient descent training (Goodfellow et al., 2016; Bishop & Bishop, 2024). It is based on the assumption that loss gradients provide a useful approximation of how the loss changes locally, which is assured by the meaningful differentiability almost everywhere and the continuity of components typically used in all kinds of ANNs. In addition, many key developments in ANN research of the last decades have addressed rugged loss landscapes (Li et al., 2018; Santurkar et al., 2018) and, related, exploding and vanishing gradients (Pascanu et al., 2013).

Spiking neuron models, by contrast, appear ill-suited for gradient descent training because they interact via discrete spikes. In the extensively used leaky integrate-and-fire (LIF) neuron model, spikes may (dis)appear at arbitrary points in time due to arbitrarily small parameter changes (Klos &

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Memmesheimer, 2025), which can induce disruptive changes of subsequent spiking dynamics and discontinuities in the loss landscape.

For exact, spike-based gradient descent, where spike times are treated as differentiable functions of the learnable parameters (Bohte et al., 2002; Eshraghian et al., 2023), the discontinuities are not predictable from gradients computed in their local neighborhood. Hence, the assumption that gradients provide a useful local approximation of the loss can fail. An often encountered consequence is the disappearance of spikes whose existence is important to achieve a small loss (Comsa et al., 2020; Göltz et al., 2021; Mostafa, 2017; Nowotny et al., 2025; Kheradpisheh & Masquelier, 2020). To counteract these consequences, most current approaches rely on ad-hoc measures (Eshraghian et al., 2023).

Recently, a conceptual advance (Klos & Memmesheimer, 2025) has offered a different, more systematic solution. It has discovered that certain neuron models, including the standard quadratic integrate-and-fire (QIF) neuron, exhibit provably continuous or even smooth spiking dynamics and thus loss landscapes. In these models, spikes cannot (dis)appear at arbitrary points in time, but only at the end of a simulation run, where such events do not disrupt subsequent activity. This property also allows the gradient-based addition of new spikes by means of so-called pseudospikes. However, it remains unclear whether this conceptual advance translates to a practical advantage. More broadly, the analysis of loss landscapes for SNNs is still in its infancy.

1.1 Contributions

We carefully compare QIF and LIF networks in a practically relevant setting using exact, spike-based gradient descent, focusing on visualization and analysis of the loss and loss gradient landscapes. Specifically, our contributions are as follows:

- We perform a thorough hyperparameter search for QIF and LIF networks with infinitesimally short input currents for the Spiking Heidelberg dataset using time-to-first-spike coding. We find that QIF networks outperform LIF networks, particularly when imposing noise-robust distinct output spike times.
- We analyze the loss landscape in the region of the training trajectories of the best-performing models. For LIF networks, the sample loss is fragmented into regions bounded by discontinuities, which are attenuated but remain present when considering batches. For QIF networks, the sample loss is less fragmented, with continuous, albeit sharp, boundaries. Averaging over samples yields a visually smooth batch loss. See surface plots in Fig. 1A-D.
- We similarly analyze gradient landscapes. For LIF networks, the sample gradients are even more fragmented and yield highly erratic batch gradients. For QIF networks, the sample gradients also contain discontinuities, but are much less fragmented, thus yielding less erratic batch gradients. See orientation maps in Fig. 1A-D.
- Zooming in, we show that for LIF networks fragment boundaries appear to coincide with disruptive spike (dis)appearances. For QIF networks, fragment boundaries are characterized by large gradients leading to rapid but continuous spike time changes. See Fig. 1E,F.
- We show that QIF networks exhibit more gradual loss changes between training steps than LIF networks, indicating that the gradient better predicts the resulting loss change.

1.2 Related work

Spike-based gradient descent. Spike-based gradient descent has recently gained popularity due to its use of exact gradients and the ability to exploit sparse communication via spikes during training and simulation (Klos & Memmesheimer, 2025; Wunderlich & Pehle, 2021; Müller et al., 2024; König et al., 2026; Comsa et al., 2020; Göltz et al., 2021; Mostafa, 2017; Nowotny et al., 2025; Kheradpisheh & Masquelier, 2020; Engelken, 2023). Most approaches use LIF or closely related neurons (Wunderlich & Pehle, 2021; Müller et al., 2024; König et al., 2026; Comsa et al., 2020; Göltz et al., 2021; Mostafa, 2017; Nowotny et al., 2025; Kheradpisheh & Masquelier, 2020; Engelken, 2023). Spike (dis)appearances are typically dealt with ad-hoc measures such as heuristic changes to input weights or spiking thresholds to revive permanently silent neurons, the introduction of activity regularization, and specialized loss functions (Eshraghian et al., 2023; Nowotny et al., 2025). Inspired

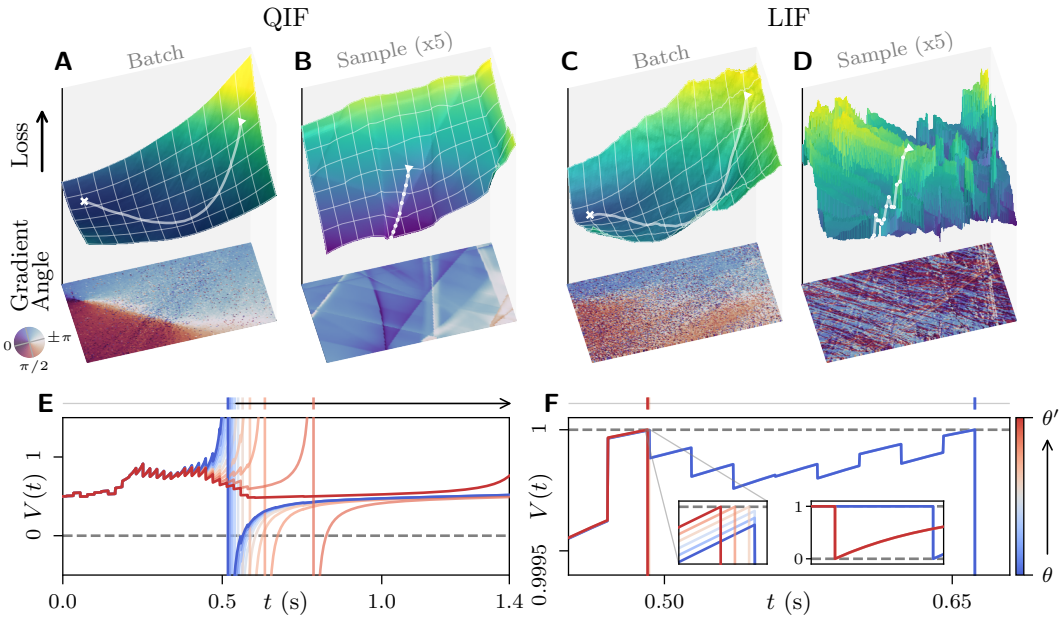

Figure 1: Loss and gradient landscapes of LIF networks are more fragmented than those of QIF networks in training-relevant parameter regions. (A,C) From a coarse-scale view, batch loss landscapes vary fairly gradually for both QIF and LIF networks, while batch gradient angle landscapes appear more noisy for LIF than QIF networks. (B,D) A fine-scale view of sample landscapes reveals discontinuities in the loss and increased fragmentation of loss and gradient landscapes for LIF networks (wireframe hidden in (D) to decrease clutter). Thick white lines indicate training trajectories (starting at triangles, points show update steps). (E,F) Across fragment boundaries, spike times of some neurons change sharply. Voltage traces (boxes) and spike times (tick marks above) are shown for two example transitions in several steps (intermediate colors) from parameters θ (blue) to θ' (red). For the QIF neuron (E), an initially early appearing spike shifts continuously until leaving the trial at its end. For the LIF neuron (F), an initially late spike changes discontinuously: a new spike appears before the second input spike upon threshold crossing (left inset), the previous spike vanishes due to the new reset (right inset).

by Klos & Memmesheimer (2025), a few studies have started using QIF neurons with indications of improved performance compared to other neuron models in the context of physics-informed neural networks (Wan et al., 2025) and the development of event-based simulation methods (König et al., 2026).

Surrogate gradient descent. A different approach to gradient-based training of SNNs is surrogate gradient descent (Neftci et al., 2019). It is based on time-discretized SNNs, where the step function, which is necessary for spike detection, is replaced by a continuous-valued surrogate during backpropagation giving rise to non-zero surrogate gradients. This smoothens the jumps in the loss landscapes but induces bias (Gygax & Zenke, 2025) and relies on timestep-based simulations, which are inept to exploit the sparsity of spiking communication.

Relation of loss and gradient properties to trainability in ANNs. Many studies have linked loss and gradient properties of ANNs to their trainability. Prominent examples have considered residual connections and batch normalization, which enable training deeper networks, and have demonstrated that these techniques smoothen loss landscapes (Li et al., 2018; Santurkar et al., 2018) and gradients (Balduzzi et al., 2017; Santurkar et al., 2018). Other notable examples have shown that flat minima lead to better generalization (Keskar et al., 2017; Hochreiter & Schmidhuber, 1997; Foret et al., 2021) and that minima of ANNs seem to be connected by a path with near constant loss (Garipov et al., 2018; Draxler et al., 2018).

Continuity of spiking dynamics in SNNs. Only few studies have considered the continuity of spiking dynamics and loss landscapes of time-continuous SNNs. Networks of purely inhibitorily coupled LIF or rapid theta neurons have been shown to exhibit phase spaces that decompose into

regions of local stability, termed flux tubes, whose boundaries are related to changes in the temporal order of spikes (Jahnke et al., 2008; Monteforte & Wolf, 2012; Engelken et al., 2024). For networks of non-leaky integrate-and-fire neurons that spike at most once, it has been shown that the input domain decomposes into so-called causal pieces, for which output spike times are locally Lipschitz continuous (Dold & Petersen, 2025). Finally, Klos & Memmesheimer (2025) have proven the continuity and even smoothness of spiking dynamics and loss landscapes in a class of neuron models that include QIF neurons.

2 METHODS

2.1 MODELS

We consider networks of oscillating LIF and QIF neurons. The subthreshold dynamics of an LIF neuron are given by

$$\tau \dot{V}_i(t) = -V_i(t) + I_i(t), \tag{1}$$

where i is the neuron index, $V_i(t)$ the membrane potential of neuron i, $I_i(t)$ the input current of neuron i and τ the membrane time constant. When $V_i(t)$ surpasses the threshold potential $V_\Theta = 1$, neuron i emits a spike and $V_i(t)$ is reset to $V_{\text{reset}} = 0$. The subthreshold dynamics of a QIF neuron are given by

$$\tau \dot{V}_i(t) = -V_i(t) + V_i^2(t) + I_i(t). \tag{2}$$

Once the voltage is large enough, the quadratic nonlinearity leads to a voltage self-amplification, analogous to the spike upstroke of biological neurons. The self-amplification is so strong that $V_i(t)$ approaches infinite voltage in finite time. Importantly, this allows to define the time of a spike t_{sp} as the time when this happens, i.e. $V_i(t_{\text{sp}}^-) = \lim_{t \to t_{\text{sp}}^-} V_i(t) = V_\Theta = \infty$. For simplicity, we call V_Θ the threshold of the QIF neuron. Afterwards, the voltage is reset to negative infinity, i.e. $V_i(t_{\text{sp}}^+) = \lim_{t \to t_{\text{sp}}^+} V_i(t) = V_{\text{reset}} = -\infty$.

For both LIF and QIF neurons, the input current is given by the sum of a constant current I_0 and infinitesimally short input currents stemming from other neurons in the network:

$$I_i(t) = I_0 + \tau \sum_j w_{ij} \sum_{t_j} \delta(t - t_j), \tag{3}$$

where j is the index of the presynaptic neurons, which spike at times t_j, and w_{ij} are the connection weights. Thus, a spike of neuron j induces an instantaneous jump by w_{ij} in the voltage of neuron i. In the case of LIF neurons, this may result in the voltage crossing the threshold potential, after which it will be immediately reset. For QIF neurons, due to the threshold being at infinity, it cannot be reached instantaneously due to an input spike. To ensure that LIF and QIF neurons oscillate intrinsically, $I_0 > V_\Theta = 1$ and $I_0 > \frac{1}{4}$, respectively, are necessary. For such neuron models, there exist simple closed-form solutions for the potential dynamics and for spike times. Networks of LIF neurons generically exhibit discontinuous spike times and spike time derivatives, while networks of QIF neurons generically exhibit continuous spike times and discontinuous spike time derivatives (Klos & Memmesheimer, 2025). To enable the systematic addition of new spikes, we continue to evolve the neurons according to so-called pseudodynamics after a trial has ended. They are only used for training and excluded for evaluation. See Sec. A for details.

2.2 SIMULATION AND GRADIENT COMPUTATION

We employ numerically precise event-based simulations (Brette et al., 2007), iterating over spikes. Briefly, in each iteration, the neuron that spikes next is identified, the corresponding spike time is computed, all neurons are advanced to this time, spike-imposed updates are applied, and the spiking neuron is reset (Sec. A.6). Event-based simulations enable the computation of exact spike-based gradients via automatic differentiation (Klos & Memmesheimer, 2025; Engelken, 2023; Müller et al., 2024; König et al., 2026) and, unlike previous approaches for the computation of numerically precise exact spike-based gradients (Comsa et al., 2020; Göltz et al., 2021; Mostafa, 2017; Kheradpisheh & Masquelier, 2020), allow for an arbitrary number of spikes per neuron.

In our simulations, the vast majority of spikes are input spikes. To avoid iterating over all of them, we bin them using periodic virtual input spikes (see code of Klos & Memmesheimer (2025)). Each virtual input spike covers the effect of all input spikes appearing in its neighborhood by summing its output weights (Sec. B.1). This computation is easily parallelized and significantly accelerates the simulation. Our code is available at `https://anonymous.4open.science/r/spikegd_evaluation-0083`.

2.3 Task

We consider the classification of spoken digits using the Spiking Heidelberg Digits (SHD) dataset (Cramer et al., 2022), which is a standard benchmark task for SNNs. We train two-layer feedforward networks with 128 hidden neurons and 20 output neurons, one for each class, using batches of 1,000 samples, which are reshuffled in every epoch. The input data is preprocessed and augmented using established methods (Sec. B).

We use time-to-first-spike coding, i.e. there is one output neuron per class and the predicted class is given by the first spiking neuron. Such coding is widely used in neuromorphic computing (Wunderlich & Pehle, 2021; Göltz et al., 2021; Cramer et al., 2022; Nowotny et al., 2025) and supported by experimental evidence from neuroscience (Gollisch & Meister, 2008; Johansson & Birznieks, 2004; Thorpe et al., 2001). For training, we use the standard cross-entropy loss with logits $-t_i^{\text{first}}/T$, where t_i^{first} is the time of the first spike of output neuron i, which may be either an ordinary spike or a pseudospike, and T is the trial length. The logits are scaled by the logit scale β, also commonly referred to as the inverse temperature. To ensure the model does not rely on pseudospikes for classification after training, we add a regularization term that pulls the first spike of the neuron coding for the correct output class forward in time in case it is a pseudospike (Sec. A).

3 Results

3.1 Model performance

For a fair comparison of LIF and QIF networks, we perform a thorough hyperparameter search. Specifically, we first perform a rough grid search and then an additional finer grid search focusing on the region with highest test accuracies (see Sec. C for details including the performance for each parameter set and Table 1 for the resulting final hyperparameters). For the best-performing QIF model, we find a final test accuracy of $90.1(9)\,\%$ after 500 epochs, compared to only $79.2(13)\,\%$ for LIF neurons (mean and std. over 10 seeds; see Fig. 2D). In addition, the LIF model exhibits much more synchronous output spikes after training compared to the QIF model (Fig. 2 A,B). This is because the hyperparameter search yields a comparatively small initial weight scale and learning rate and a large logit scale for the LIF model, resulting in a small initial temporal spread of the output spikes and small changes of output spike times during training.

Given that almost synchronous output spikes are highly susceptible to noise, which is present in many neuromorphic hardware systems (Roy et al., 2019), we also seek an LIF model with asynchronous output spikes. To this end, we measure the difference between the first spike times of the output neurons that spike first and second, averaged over all samples in the training set. We call this distinctness t_{dis} (Fig. 2E). The best-performing LIF model has a low average distinctness of $3\,\text{ms}$ in the final epoch (averaged over samples and seeds), while the best-performing QIF model has one of $28\,\text{ms}$. There is only a single LIF hyperparameter configuration in the fine search with an average distinctness above $20\,\text{ms}$, which we select for the asynchronous model (Fig. 2C, hyperparameters in Table 1) and also train again for 500 epochs and 10 seeds. While having an average distinctness of $29\,\text{ms}$ in the final epoch, comparable to the best QIF model, this LIF network only achieves about $50.8(40)\,\%$ test accuracy (Fig. 2D,E).

In additional exploratory experiments with larger networks, we find an improved final test accuracy of $91.3(8)\,\%$ for a QIF and $85.2(13)\,\%$ for an LIF network (Sec. D). While these results are still markedly below the state-of-the-art of about 96% (Sun et al., 2025; Schöne et al., 2024; Baronig et al., 2025), we note that we use comparably simple network architectures and neurons and focus on comparing LIF and QIF neurons rather than maximizing accuracy.

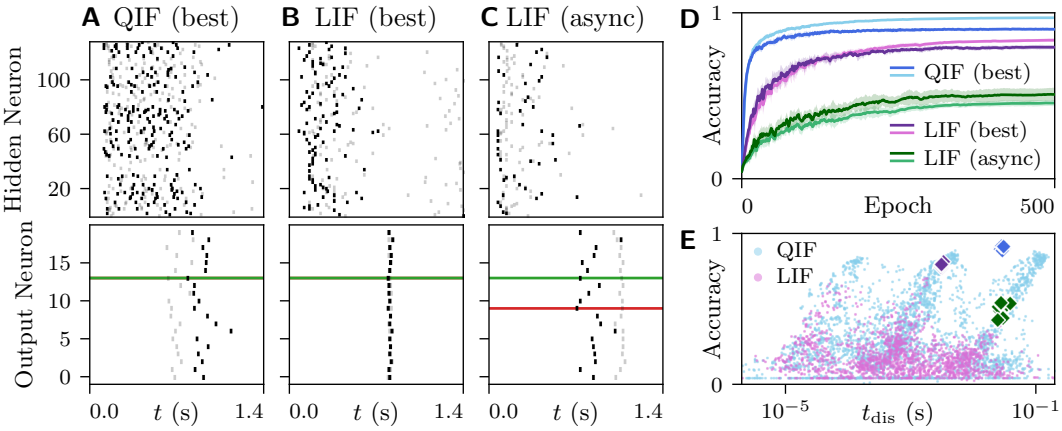

Figure 2: Comparison of the three selected models, the best-performing QIF model, the best-performing LIF model and the asynchronous LIF model. (A-C) Example spike trains before (gray) and after (black) training. True label indicated by green line, model prediction by red line, if different from true label. (D) Accuracy during training for the training (brighter colors) and test (darker colors) datasets. Solid lines indicate mean and shaded area standard deviation over 10 seeds. (E) Test accuracy vs. average distinctness in final epoch of the rough hyperparameter search configurations. The three selected models are indicated by diamonds colored as in D. Only QIF networks achieve both high distinctness and high accuracy.

3.2 Loss and Gradient Landscapes

To investigate the origin of the performance difference between QIF and LIF models, we visualize the loss and gradient landscapes of all three selected models. Due to the high dimensionality of the parameter space, it is impossible to visualize the landscapes in their entirety. We thus select a slice of a two-dimensional plane in the parameter space on which we evaluate the model. We note that the position, orientation and scale of the selected slice must be chosen carefully, because it strongly affects the resulting landscapes (Li et al., 2018).

We focus on the region relevant for training and use a modified version of the slice selection proposed by Li et al. (2018). Specifically, we select the primary direction u of the plane to be the unit vector pointing from the initial parameters θ_0 to the final parameters θ_K (after 200 epochs), where $K = 1{,}800$ is the number of update steps. For the secondary direction v, we select the first principal component of the vectors $[\theta_1 - \theta_0, \ldots, \theta_{K-1} - \theta_0]$ projected onto the subspace orthogonal to u (Sec. F.1). We parametrize the points on the plane using two coordinates (α, β), where α varies along u and β varies along v. The parameters at (α, β) are then given by

$$\theta(\alpha, \beta) = \theta_0 + \alpha s u + \beta s v, \tag{4}$$

where $s = |\theta_K - \theta_0|$ is a scaling factor. Thus, unlike the approach used by Li et al. (2018), both the initial parameters θ_0 and the final parameters θ_K lie on the plane, at coordinates $(0, 0)$ and $(1, 0)$, respectively.

We start by taking on a large-scale perspective on the entire training trajectory (Fig. 1A,C and Fig. 3, Fig. 16 shows additional examples and gradient amplitude landscapes). For all three models, the loss landscapes for a single input sample appear fragmented with gradually changing loss within and sharp loss changes on the boundaries of fragments (Fig. 3A). This is confirmed quantitatively by the larger correlation length of the QIF landscapes compared to both LIF landscapes (QIF: $0.0235(30)$, LIF (best): $0.0152(18)$, LIF (async): $0.0153(21)$; mean and std. over 16 samples and 5 seeds; see Sec. F.2 for details). We also observe many coinciding fragment boundaries in the loss and the angle of the loss gradients $(\frac{\partial L}{\partial \alpha}, \frac{\partial L}{\partial \beta})$ for the same sample (Fig. 3B). For QIF neurons, we again observe gradually changing angles within and sharp changes on the boundaries of fragments. For LIF neurons, however, it appears as if the fragmentation is overlaid with noise, leading to erratically changing gradient directions also within fragments. This manifests in a larger cosine similarity (QIF: $0.862(32)$, LIF (best): $0.309(53)$, LIF (async): $0.651(46)$; see Sec. F.2 for details).

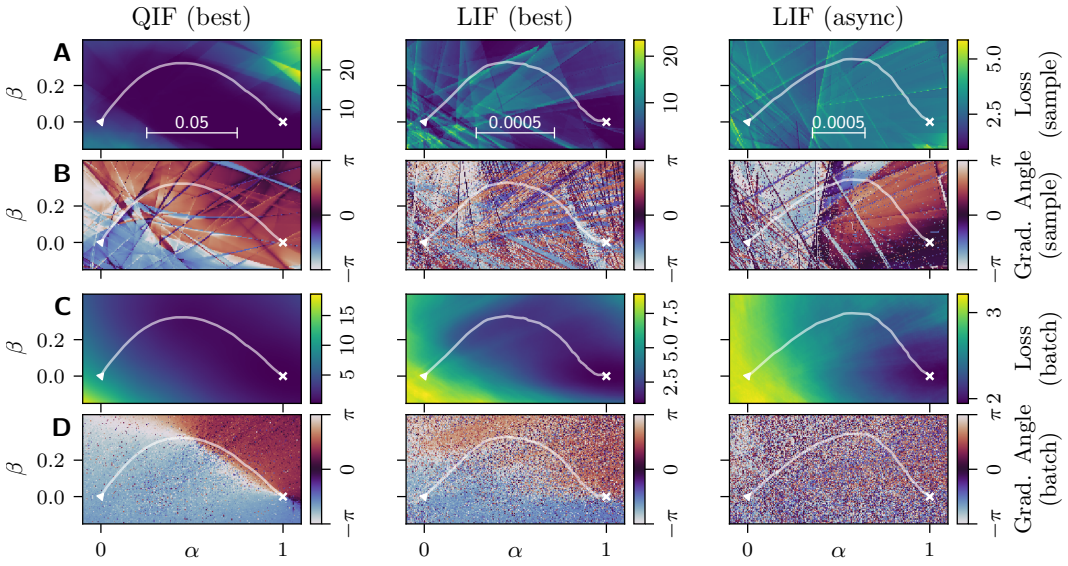

Figure 3: Loss and gradient landscapes for three selected models. (A) Sample loss landscapes for an example input sample. (B) Gradient angle landscape corresponding to the sample loss shown in A. (C) Batch loss landscapes for an example input batch containing 1,000 samples. (D) Gradient angle landscape corresponding to the batch loss shown in C. White lines indicate training trajectories over 200 epochs (starting at triangles). Scale bars in A show distances in parameter space.

Since training uses batches rather than individual samples, we also examine batch loss and gradient landscapes. Given the large batch size, we expect them to be qualitatively independent of the specific batch and thus only consider a single one. Averaging over the samples in a batch substantially smoothens the loss landscapes: fragmentation is no longer clearly visible (Fig. 3C) and correlation lengths increase (QIF: $0.110(25)$, LIF (best): $0.046(8)$, LIF (async): $0.032(6)$; mean and std. over 5 seeds). In contrast, while the batch gradient landscapes likewise no longer exhibit clearly delineated macroscopic fragments, they appear much noisier, in particular for LIF neurons with asynchronous output spikes and, to a lesser degree, for the best-performing LIF network (Fig. 3D and Fig. 16). This is reflected by the different cosine similarities (QIF: $0.818(29)$, LIF (best): $0.417(24)$, LIF (async): $0.033(10)$). Thus, the roughness differences of the batch loss and gradient landscapes follow the performance ranking of the three models.

Together, these results suggest smoother loss and gradient landscapes as a reason for the better performance of QIF networks.

3.3 Discontinuities in loss and gradient

To examine the cause of the loss fragmentation and the, on a large-scale, erratic gradients, we visualize the landscapes on a much finer scale. Zooming in on the sample landscapes reveals that the fragments visible in Fig. 3A consist of many smaller fragments (Fig. 1B,D and Fig. 17). For LIF neurons, the loss fragments are separated by discontinuous jumps and form plateaus with slowly varying loss. The gradient angles are fragmented even further into a similar plateau-like structure.

This fine-grained fragmentation and the discontinuities in both loss and gradient become particularly clear when focusing on a small part of a single update step (Fig. 4). The depicted example shows a discontinuous jump in the loss (arrow a in Fig. 4), which is accompanied by a small discontinuous jump in the gradient direction (inset in Fig. 4D). Examining the network dynamics reveals that it results from the disappearance of a hidden-layer spike at one point in time and the appearance of a new spike of that same neuron at an earlier point in time (Fig. 1F). This (dis)appearance event is caused by a small change in the input weights, which results in the membrane potential reaching the threshold directly before the next, inhibitory (virtual) input spike arrives, which previously prevented the new output spike. In other words, the spike (dis)appearance occurs when an input and output spike time change their temporal order, as predicted from theory (Klos & Memmesheimer, 2025).

The example also contains a discontinuity in the gradient direction without an accompanying discontinuity in the loss or any spike times (arrow b in Fig. 4). It results from a change in how a hidden neuron reaches its threshold. Before the discontinuity, a virtual input spike pushes the membrane potential above the threshold, leading to a locally constant spike time. Afterwards, due to the parameter changes, the neuron already reaches the threshold by itself because of its suprathreshold input drive, leading to a spike time with nonzero gradient (Sec. G). While this mechanism is only present for inputs causing voltage jumps, gradient discontinuities generically always occur when spikes (dis)appear, including for LIF neurons with exponentially decaying input currents, i.e. for inputs causing only input current jumps (Klos & Memmesheimer, 2025).

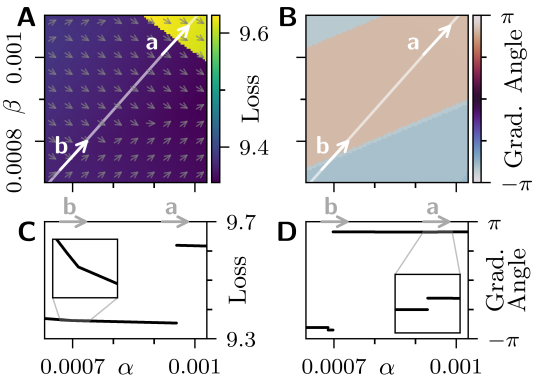

Figure 4: Example loss and gradient discontinuities in LIF neurons. (A) Sample loss landscape zoomed into a region containing both a loss discontinuity (arrow a) and a (pure) gradient discontinuity (arrow b). Gray arrows indicate direction of negative gradient. White line indicates training trajectory. (B) Gradient angle landscape of sample loss shown in (A). (C, D) Sample loss and gradient angle along the training trajectory in the same region of α coordinates as in (A, B).

In contrast, we found no loss discontinuities for QIF neurons, consistent with theoretical results (Klos & Memmesheimer, 2025). The sharp boundaries observable in Fig. 3A are, instead, continuous transitions with large gradient amplitudes and hidden neuron spike time shifts (Sec. H, Fig. 1E). While such events lead to changes in the temporal order of spikes, they do not result in spike (dis)appearances since the effect of input spikes vanishes in the vicinity of an output spike as a result of the divergence of the QIF membrane potential. In contrast, the gradient exhibits discontinuities. They occur when the order of two hidden neuron spikes that affect the loss changes. This is because the derivative of an output spike time with respect to an input spike time jumps when this input spike crosses another input spike. However, given that the jump magnitude is very small for all crossings we examined (Sec. I), we do not expect them to have a notable effect on network training. The mechanism underlying the gradient discontinuity described in the previous paragraph does not apply to QIF neurons because input spikes cannot force an immediate output spike. This may be a reason why the fragmentation of the gradient landscape is not as severe as for LIF neurons.

In summary, the fragmentation of the loss and gradient landscapes results from disruptive spike (dis)appearances and changes in the temporal order of spikes for LIF neurons and from large gradients for QIF neurons. The fragmentation is more severe for LIF neurons. The noise-like appearance of some of the large-scale landscapes results from the inability to resolve this fine-grained fragmentation.

3.4 Effect of discontinuities on training

The results of the landscape analysis suggest the existence of loss discontinuities and the increased fragmentation as reasons for the worse performance of LIF neurons. To strengthen this link, we next follow the training of the three selected models along their actual trajectories through the high-dimensional parameter space, which differ from their projection onto the two-dimensional planes shown in the previous section. First, we track how the loss of the batch used in training step k changes along the linear path between the parameters θ_k and θ_{k+1}, before and after the update for one example training run per model (Fig. 5A). We observe that the loss changes much more gradually for QIF than for LIF neurons. As observed in the loss landscapes, there are further many discontinuities for the LIF neurons, in particular in the asynchronous model. Consistent with these observations, we find that for the QIF network $90.0(11)\,\%$ of the training steps for which a loss decrease is expected from the gradient in fact results in a decrease, while this fraction is only $73.4(12)\,\%$ for the best-performing LIF network and $61.3(19)\,\%$ for the asynchronous LIF network (mean and std. over 10 seeds; see Sec. K for details). Thus, the gradient is more informative for training in QIF networks.

Finally, motivated by our observation of input spikes to LIF neurons that do not contribute to the gradient (Sec. 3.3), we track the fraction r of gradient components that are zero for all samples within

an epoch (Fig. 5B). Larger fractions imply fewer active parameters and hence a potentially reduced expressivity. We find that this fraction is $62.2(47)\,\%$ at the end of training for LIF neurons with asynchronous output spikes, but only $26.0(20)\,\%$ and $26.1(48)\,\%$ for the best-performing LIF and QIF models, respectively (mean and std. over 10 seeds and the last 20 epochs). We also notice that this fraction abruptly decreases (and increases again) several times during training for the asynchronous LIF model. This happens when a spike of an output neuron disruptively disappears (and reappears again) during the trial and, for the loss computation, is replaced by a pseudospike, which depends on all hidden layer neurons. These results suggest that reduced expressivity may explain the poor performance of LIF neurons with asynchronous output spikes, but not the remaining performance gap between the best-performing LIF and QIF models.

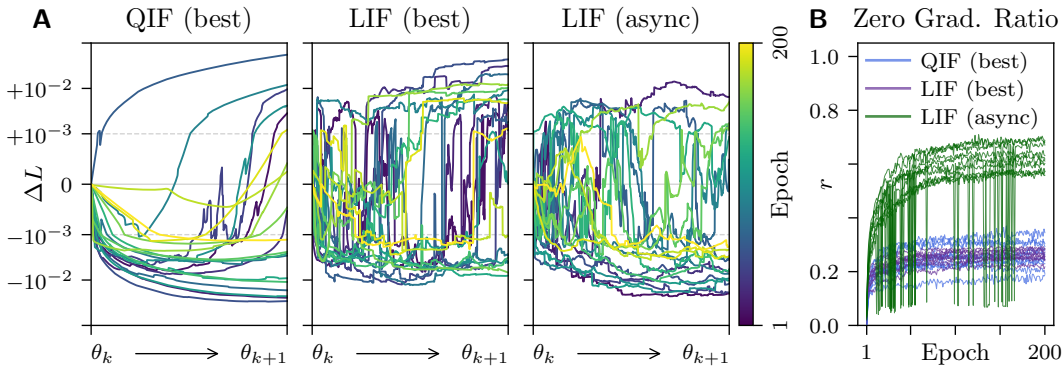

Figure 5: Analysis of the training trajectory. (A) Loss change along linear path between parameters θ_k and θ_{k+1} corresponding to the start and end of training step k. To reduce clutter, only the first update step of every 10-th epoch is shown (20 steps in total), each in a different color. The loss change $\Delta L(\theta) = L(\theta) - L(\theta_k)$ is shown using a symmetric log scale with a linear region between $\pm 10^{-3}$, indicated by dashed gray lines. Negative ΔL indicate a performance improvement and are thus more common in well-performing models. (B) Fraction r of gradient components that vanish for all samples within each epoch. One line per model and seed (10 seeds per model).

4 Discussion

We have shown that networks of QIF neurons outperform networks of LIF neurons on the SHD dataset and visualized their loss and gradient landscapes. We have found that the landscapes are strikingly fragmented. For LIF networks, the fragmentation is more severe and fragments are separated by discontinuities for both the loss and gradient. For QIF networks, the fragment boundaries are characterized by large gradients with sharply but continuously changing loss. Consistent with the performance difference, this leads to a more rugged loss along the training path for LIF compared to QIF networks. Thus, the guaranteed continuity of spiking dynamics in QIF networks appears to translate into an advantage in practice.

We have focused on a single setting, i.e. in particular a single task and output coding scheme. Further, we have only considered intrinsically oscillating neurons and input currents that cause jumps directly in the membrane potential and not the more popular exponentially decaying input currents. This may seem restrictive, but has allowed us to perform a thorough hyperparameter search for both QIF and LIF neurons, which we deem necessary for a fair comparison, and to analyze the loss and gradient landscapes in detail. Previous work has achieved competitive performance with LIF neurons using spike-based gradient descent in different settings. However, this often requires modifying the loss function (Nowotny et al., 2025) or using careful weight initialization and ad-hoc methods to counteract silent neurons (Eshraghian et al., 2023). Together with the conceptual advantage of guaranteed continuity, our empirical results indicate that QIF neurons are more robust to the specifics of the considered setup. Nevertheless, an important task for future research is to test if our results extend to other settings as well. It would also be interesting to test if QIF neurons are advantageous when using surrogate instead of spike-based gradient descent.

In conclusion, our work suggests to replace the ubiquitously used LIF neurons with neurons exhibiting continuous spiking dynamics, such as QIF neurons, when training SNNs with gradient descent.

AI USE STATEMENT

In this work, we used generative AI tools to design or provide feedback on research methodology or experiments and assist with translation. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, implement methods, clean and reformat datasets, support qualitative and thematic data analysis, and interpret results, and assistance in the writing of proofs is not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, summarize or analyse existing literature, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature, format references, and propose a title or keywords for a research paper. We have reviewed all AI-assisted work. LLM-generated code was manually verified and tested for correctness. LLM-formatted references were manually checked. LLM feedback on research methods, such as landscape creation and roughness measures, was critically checked and the relevant literature was manually verified and cited. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All main results of this paper are reproducible using the provided code at `https://anonymous.4open.science/r/spikegd_evaluation-0083`. Instructions for running the experiments and analyses are given in the README file. The methods and data processing steps are described in detail in Sec. 2, Sec. A, and Sec. B. Estimations of required compute resources are given in Sec. L.

REFERENCES

David Balduzzi, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: if resnets are the answer, then what is the question? In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 342–350. PMLR, 2017.

Maximilian Baronig, Romain Ferrand, Silvester Sabathiel, and Robert Legenstein. Advancing Spatio-Temporal Processing in Spiking Neural Networks through Adaptation, 2025.

Christopher M. Bishop and Hugh Bishop. *Deep Learning - Foundations and Concepts*. Springer, 2024. ISBN 978-3-031-45467-7.

Alexandre Bittar and Philip N. Garner. A surrogate gradient spiking baseline for speech command recognition. *Frontiers in Neuroscience*, 16:865897, 2022. doi: 10.3389/fnins.2022.865897.

Sander M. Bohte, Joost N. Kok, and Han La Poutré. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1):17–37, 2002. doi: 10.1016/S0925-2312(01)00658-0.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M. Bower, Markus Diesmann, Abigail Morrison, Philip H. Goodman, Frederick C. Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P. Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, 2007. doi: 10.1007/s10827-007-0038-6.

Iulia M. Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function: Learning with Backpropagation. In *ICASSP 2020 – 2020 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8529–8533. IEEE, 2020. doi: 10.1109/ICASSP40776.2020.9053856.

Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2744–2757, 2022. doi: 10.1109/TNNLS.2020.3044364.

Peter Dayan and L. F. Abbott. *Theoretical Neuroscience*. MIT Press, Cambridge, 2001. ISBN 0-262-04199-5.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020.

Dominik Dold and Philipp Christian Petersen. Causal pieces: analysing and improving spiking neural networks piece by piece, 2025.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A. Hamprecht. Essentially no barriers in neural network energy landscape. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1309–1318. PMLR, 2018.

Rainer Engelken. SparseProp: Efficient Event-Based Simulation and Training of Sparse Recurrent Spiking Neural Networks. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Rainer Engelken, Michael Monteforte, and Fred Wolf. Sparse chaos in cortical circuits, 2024.

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. doi: 10.1109/JPROC.2023.3308088.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-Aware Minimization for Efficiently Improving Generalization. In *International Conference on Learning Representations*, 2021.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. ISBN 978-1-107-06083-8.

Tim Gollisch and Markus Meister. Rapid Neural Coding in the Retina with Relative Spike Latencies. *Science*, 319(5866):1108–1111, 2008. doi: 10.1126/science.1149639.

J. Göltz, L. Kriener, A. Baumbach, S. Billaudelle, O. Breitwieser, B. Cramer, D. Dold, A. F. Kungl, W. Senn, J. Schemmel, K. Meier, and M. A. Petrovici. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature Machine Intelligence*, 3(9):823–835, 2021. doi: 10.1038/s42256-021-00388-x.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Julia Gygax and Friedemann Zenke. Elucidating the theoretical underpinnings of surrogate gradient learning in spiking neural networks. *Neural Computation*, 37(5):886–925, 2025. doi: 10.1162/neco_a_01752.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015. doi: 10.1109/iccv.2015.123.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997. doi: 10.1162/neco.1997.9.1.1.

J. D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.

Sven Jahnke, Raoul-Martin Memmesheimer, and Marc Timme. Stable irregular dynamics in complex neural networks. *Physical Review Letters*, 100(4):048102, 2008. doi: 10.1103/PhysRevLett.100.048102.

Roland S. Johansson and Ingvars Birznieks. First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature Neuroscience*, 7(2):170–177, 2004. doi: 10.1038/nn1177.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *International Conference on Learning Representations*, 2017.

Saeed Reza Kheradpisheh and Timothée Masquelier. S4NN: temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, 30(06): 2050027, 2020. doi: 10.1142/S0129065720500276.

Christian Klos and Raoul-Martin Memmesheimer. Smooth Exact Gradient Descent Learning in Spiking Neural Networks. *Physical Review Letters*, 134(2):027301, 2025. doi: 10.1103/PhysRevLett.134.027301.

Lukas König, Manuel Kuhn, David Kappel, and Anand Subramoney. Training event-based neural networks with exact gradients via Differentiable ODE Solving in JAX, 2026.

Dhireesha Kudithipudi, Catherine Schuman, Craig M. Vineyard, Tej Pandit, Cory Merkel, Rajkumar Kubendran, James B. Aimone, Garrick Orchard, Christian Mayr, Ryad Benosman, Joe Hays, Cliff Young, Chiara Bartolozzi, Amitava Majumdar, Suma George Cardwell, Melika Payvand, Sonia Buckley, Shruti Kulkarni, Hector A. Gonzalez, Gert Cauwenberghs, Chetan Singh Thakur, Anand Subramoney, and Steve Furber. Neuromorphic computing at scale. *Nature*, 637(8047):801–812, 2025. doi: 10.1038/s41586-024-08253-8.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Wes McKinney. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pp. 51–56, 2010. doi: 10.25080/Majora-92bf1922-00a.

A. Mehonic and A. J. Kenyon. Brain-inspired computing needs a master plan. *Nature*, 604(7905): 255–260, 2022. doi: 10.1038/s41586-021-04362-w.

Michael Monteforte and Fred Wolf. Dynamic Flux Tubes Form Reservoirs of Stability in Neuronal Circuits. *Physical Review X*, 2(4):041007, 2012. doi: 10.1103/PhysRevX.2.041007.

Hesham Mostafa. Supervised Learning Based on Temporal Coding in Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–9, 2017. doi: 10.1109/TNNLS.2017.2726060.

Eric Müller, Moritz Althaus, Elias Arnold, Philipp Spilger, Christian Pehle, and Johannes Schemmel. jaxsnn: Event-driven Gradient Estimation for Analog Neuromorphic Hardware, 2024.

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. doi: 10.1109/MSP.2019.2931595.

Thomas Nowotny, James P Turner, and James C Knight. Loss shaping enhances exact gradient learning with Eventprop in spiking neural networks. *Neuromorphic Computing and Engineering*, 5(1):014001, 2025. doi: 10.1088/2634-4386/ada852.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pp. 1310–1318. PMLR, 2013.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019. doi: 10.1038/s41586-019-1677-2.

Hannes P Saal, Xiaoqin Wang, and Sliman J Bensmaia. Importance of spike timing in touch: an analogy with hearing? *Current Opinion in Neurobiology*, 40:142–149, 2016. doi: 10.1016/j.conb.2016.07.013.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How Does Batch Normalization Help Optimization? In *Advances in Neural Information Processing Systems*, volume 31, pp. 2488–2498. Curran Associates, Inc., 2018.

Mark Schöne, Neeraj Mohan Sushma, Jingyue Zhuge, Christian Mayr, Anand Subramoney, and David Kappel. Scalable Event-by-event Processing of Neuromorphic Sensory Signals With Deep State-Space Models, 2024.

Samuel J. Sober, Simon Sponberg, Ilya Nemenman, and Lena H. Ting. Millisecond Spike Timing Codes for Motor Control. *Trends in Neurosciences*, 41(10):644–648, 2018. doi: 10.1016/j.tins.2018.08.010.

Pengfei Sun, Jibin Wu, Paul Devos, and Dick Botteldooren. Towards parameter-free attentional spiking neural networks. *Neural Networks*, 185:107154, 2025. doi: 10.1016/j.neunet.2025.107154.

The pandas development team. pandas-dev/pandas: Pandas, 2020.

Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14(6):715–725, 2001. doi: 10.1016/S0893-6080(01)00083-1.

Ruyin Wan, George Em Karniadakis, and Panos Stinis. From LIF to QIF: Toward Differentiable Spiking Neurons for Scientific Machine Learning, 2025.

Jason Wolfe, Arthur R Houweling, and Michael Brecht. Sparse and powerful cortical spikes. *Current Opinion in Neurobiology*, 20(3):306–312, 2010. doi: 10.1016/j.conb.2010.03.006.

Timo C. Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):12829, 2021. doi: 10.1038/s41598-021-91786-z.

Weizhen Xie, John H. Wittig, Julio I. Chapeton, Mostafa El-Kalliny, Samantha N. Jackson, Sara K. Inati, and Kareem A. Zaghloul. Neuronal sequences in population bursts encode information in human cortex. *Nature*, 635(8040):935–942, 2024. doi: 10.1038/s41586-024-08075-8.

A MODEL DETAILS

The model description was intentionally kept short in the main text to keep the focus on the main results. For completeness, we give more details about the models used in this work in this section. This is intended to be an extension of Sec. 2.1.

A.1 PHASE REPRESENTATION

To simplify and unify the dynamics of oscillating LIF and QIF neuron models, we transform the membrane potential $V(t)$ into a phase variable $\phi(t)$ using a phase transformation $\Phi(V) = \phi$. We require the phase variable to increase linearly with time between spikes, according to the simple dynamical equation $\dot{\phi} = 1$, and to be reset to $\phi_{\text{reset}} = 0$ after reaching the threshold phase $\phi_\Theta = \lim_{V \to V_\Theta} \Phi(V)$. The following phase transformations fulfill these properties,

$$\Phi^{\text{LIF}}(V) = -\tau \log\left(1 - \frac{V}{I_0}\right), \qquad \Phi^{\text{QIF}}(V) = \frac{\tau}{a}\left[\arctan\left(\frac{V - \frac{1}{2}}{a}\right) + \frac{\pi}{2}\right], \tag{5}$$

where $a = \sqrt{I_0 - \frac{1}{4}}$. The resulting threshold phases are

$$\phi_\Theta^{\text{LIF}} = -\tau \log\left(1 - \frac{V_\Theta}{I_0}\right), \qquad \phi_\Theta^{\text{QIF}} = \frac{\pi\tau}{a}. \tag{6}$$

The inverse phase transformations are

$$(\Phi^{\text{LIF}})^{-1}(\phi) = I_0\left(1 - e^{-\phi/\tau}\right), \qquad (\Phi^{\text{QIF}})^{-1}(\phi) = \frac{1}{2} + a\tan\left(\frac{a\phi}{\tau} - \frac{\pi}{2}\right). \tag{7}$$

For infinitesimally short input currents, the membrane potential of neuron i jumps by w_{ij} when receiving an input spike from neuron j. The effect of this potential jump on the phase variable can be quantified using the phase transfer curve

$$H_w(\phi) = \Phi(\Phi^{-1}(\phi) + w). \tag{8}$$

The phase transfer curves of LIF and QIF neurons are

$$H_w^{\text{LIF}}(\phi) = -\tau \log\left(e^{-\phi/\tau} - \frac{w}{I_0}\right), \quad H_w^{\text{QIF}}(\phi) = \frac{\tau}{a}\left\{\arctan\left[\tan\left(\frac{a\phi}{\tau} - \frac{\pi}{2}\right) + \frac{w}{a}\right] + \frac{\pi}{2}\right\}. \tag{9}$$

Although we never show phase values directly in the main text, every network used in this work is simulated in phase representation. To reduce complexity and increase accessibility of the main text, we convert all phases into membrane potentials using the inverse transformations Φ^{-1}.

A.2 IMMEDIATE FIRING

Since LIF neurons have a finite threshold potential V_Θ, they can immediately fire when receiving an input spike from a neuron with a sufficiently large synaptic weight. The membrane potential of neuron i immediately after receiving an input from neuron j thus either jumps by w_{ij} or is reset to the reset potential $V_{\text{reset}} = 0$. Specifically, the potential right after the input spike arrived is given by

$$V_i^+ = \begin{cases} V_i^- + w_{ij} & \text{if } V_i^- + w_{ij} < V_\Theta \\ 0 & \text{if } V_i^- + w_{ij} \geq V_\Theta \end{cases}, \tag{10}$$

where V_i^- is the potential right before the input spike arrives. In phase representation (Sec. A.1), we have

$$\phi_i^+ = \begin{cases} H_{w_{ij}}(\phi_i^-) & \text{if } H_{w_{ij}}(\phi_i^-) < \phi_\Theta \\ 0 & \text{if } H_{w_{ij}}(\phi_i^-) \geq \phi_\Theta \end{cases}. \tag{11}$$

Note that if an LIF neuron fires immediately after an input spike, the resulting phase ϕ_i^+ (or membrane potential V_i^+) does not depend on the synaptic weight w_{ij}. If neuron i does not receive any other input spikes from neuron j before it has reached the threshold, the resulting spike time of neuron i does not depend on w_{ij}, leading to a vanishing gradient. QIF neurons do not have this problem since they have an infinite threshold potential, which can never be reached immediately due to an input spike.

A.3 PSEUDODYNAMICS

To enable the addition of new spikes via spike-based gradient descent, we extend a trial using so-called pseudodynamics, which guarantee that the neurons spike as often as necessary for the considered task (Klos & Memmesheimer, 2025).

The pseudodynamics are comprised of the following steps. First, at the beginning of the pseudodynamics (the trial end T), an interaction between all connected network neurons is enforced. The neurons in the first layer do not receive input from other network neurons. Hence, their voltage is equal to voltage at the end of the ordinary dynamics, i.e. $V_{\mathrm{ps},i}^{(1)}(T) = V_i^{(1)}(T)$, where $V_{\mathrm{ps},i}^{(1)}(T)$ is the voltage at the start of the pseudodynamics with the superscript indexing the layer number. For all other layers l, the voltage is

$$V_{\mathrm{ps},i}^{(l)}(T) = V_i^{(l)}(T) + \sum_j w_{ij}^{(l)} \frac{\Phi(V_{\mathrm{ps},j}^{(l-1)}(T))}{\phi_\Theta}, \tag{12}$$

where $w_{ij}^{(l)}$ are the weights of layer l. Afterwards, the neurons continue to evolve according to the same dynamics as during the ordinary dynamics (Eqs. (1) to (3)), but excluding any further interaction between the neurons. The pseudynamics are continued until all spikes necessary for the task at hand are generated. In our case this is the case when all output neurons that have not spiked during the trial have spiked once during the pseudodynamics. The first pseudospike time $t_{\mathrm{ps},i}^{(L)}$ of a neuron i in the output layer L is given by

$$t_{\mathrm{ps},i}^{(L)} = T + \phi_\Theta - H_{w_{\mathrm{ps},i}^{(L)}}(\phi_i^{\mathrm{out}}(T)), \quad w_{\mathrm{ps},i}^{(L)} = \sum_j w_{ij}^{(L)} \frac{\Phi(V_{\mathrm{ps},j}^{(L-1)}(T))}{\phi_\Theta}. \tag{13}$$

The interaction between neurons was introduced in Klos & Memmesheimer (2025) to ensure that for QIF neurons a spike time is continuous when it crosses the trial end, i.e. when it transitions from an ordinary spike to a pseudospike or vice versa (Klos & Memmesheimer, 2025), and to enable the flow of gradients to all layers via output and hidden neurons that do not spike within the trial. The computation of pseudospike times is effectively the same as a single forward pass through a feed-forward ANN and does not necessitate iterating over spikes or timesteps as in event- or timestep-based simulations.

A.4 LOSS

As usual for classification tasks, we use the cross entropy loss. The sample loss for an input sample x with corresponding true label l is

$$L_{(x,l)}(\theta) = -\log \frac{e^{\beta z_l(x,\theta)}}{\sum_j e^{\beta z_j(x,\theta)}}, \tag{14}$$

where $z_i(x,\theta)$ is the logit of the i-th output neuron and β is a scaling factor. Due to its interpretation in statistical physics, β is often referred to as the inverse temperature. In this work, we call it the logit scale to emphasize its scaling effect on the logits z_i. When training, we always evaluate the loss on a batch B containing $N_B = 1000$ input samples with labels (x, l). The mean loss across all batch samples is the batch loss

$$L_B(\theta) = \frac{1}{N_B} \sum_{(x,l)\in B} L_{(x,l)}(\theta). \tag{15}$$

The model prediction for a single input sample x is the output index i that has the largest corresponding logit z_i. In this work, we focus on time-to-first-spike coding, which assigns larger logits to output neurons that spike earlier, according to

$$z_i(x,\theta) = -\frac{t_i(x,\theta)}{T}. \tag{16}$$

Here, $t_i(x,\theta)$ denotes the first spike time of output neuron i, given the input sample x and the model parameters θ, and T is the trial duration, which serves as a normalization factor. The predicted label is thus the index of the output neuron that spikes first.

Although pseudodynamics are useful during model training, we would like our final trained model to perform well without them during inference. On the one hand, this reduces the complexity of the final model and, on the other hand, should allow for an easier implementation on neuromorphic hardware. To make sure the model does not rely on pseudospikes, we introduce a regularization term, which punishes pseudospikes that are required for inference,

$$L^{\text{reg}}_{(x,l)}(\theta) = \eta \Big[\min\left(z_l(x,\theta) - \tilde{z}, 0\right)\Big]^2. \tag{17}$$

Here, $\eta = 10^3$ is the regularization strength and $\tilde{z} = -0.98$ is the target logit. If the logit is larger than this target logit, it does not depend on pseudospikes and thus is not punished. The offset of 0.02, compared to the logit -1 of a spike at the trial end, is included to ensure that the spike time is safely inside the trial, even if small variations occur. We square the distance to get a well-defined gradient at $z_l(x,\theta) = \tilde{z}$.

A.5 Initialization

The weights of network layer l are initialized using a uniform Kaiming initialization (He et al., 2015), as was done in Cramer et al. (2022),

$$w_{ij}^{(l)} \sim \mathcal{U}\left(-\frac{w_0}{\sqrt{N_{l-1}}}, \frac{w_0}{\sqrt{N_{l-1}}}\right).$$

Here, w_0 is a hyperparameter that can be used to scale the initial weights and N_{l-1} is the number of neurons in layer $l-1$, i.e., the number of inputs to layer l. The neuron phase variables ϕ_i are initialized at $\frac{\phi_\Theta}{2}$. We also experimented with learnable and random phase initializations but did not find a significant advantage in ad-hoc experiments.

A.6 Event-based simulation

We simulate each network layer in an event-based manner, i.e., by processing each spike one-by-one and evolving the dynamics between spikes analytically. Since we only consider feedforward networks, each layer can be simulated separately. Each simulation step, indexed by k, starts at the previous spike time t_k, or at the initial time $t_0 = 0$. The phase of neuron i at the beginning of step k is denoted by ϕ_{ik}. The simulation of step k then involves the following operations:

1. Determine the neuron index j_{k+1} and spike time t_{k+1} of the next spike. This could either be an input spike or a spike in the layer that is simulated, depending on which appears earlier. Because the phase grows linearly with time between spikes ($\dot{\phi}_i = 1$), the neuron that spikes next is always the one with the largest phase, $j_{k+1} = \text{argmax}_i \phi_{ik}$.
2. Evolve neurons freely until shortly before the next spike at t^-_{k+1}. This is trivial due to the linear dynamics between spikes. We have $\dot{\phi}_i = 1$ and, thus, $\phi_i(t^-_{k+1}) = \phi_{ik} + (t_{k+1} - t_k)$.
3. If the next spike is an input spike, transfer it to the neurons in the simulated layer, using the phase transfer function $\phi_{i,k+1} = H_{w_{ij_{k+1}}}(\phi_i(t^-_{k+1}))$. Otherwise, if the spike occurs in the simulated layer itself, reset the phase of the spiking neuron, $\phi_{j+1,k+1} = 0$, and set $\phi_{i,k+1} = \phi_i(t^-_{k+1})$ for all other neurons.

This process has to be repeated until the simulation reaches the trial end T, i.e., until $t_{k+1} \geq T$.

B Data preprocessing and augmentation

To reduce computational complexity and to improve the generalization abilities of our models, we use several data preprocessing and augmentation strategies.

B.1 Temporal binning

The SHD samples contain between 2,410 and 16,257 spikes. To avoid iterating over all of them, which is computationally inefficient, we discretize the trial duration T into N_T bins. Bin $k = 1, \ldots, N_\text{T}$

covers the range $\frac{(k-1)T}{N_\mathrm{T}} \leq t_j < \frac{kT}{N_\mathrm{T}}$. Then, we replace all input spikes within each bin k by a single virtual input spike located at the bin center,

$$t_k^{\text{virt}} = \frac{\left(k - \frac{1}{2}\right) T}{N_\mathrm{T}}. \tag{18}$$

We use $T = 1.4\,\mathrm{s}$, which fully covers the SHD spike-time range (last input spike at $1.369\,\mathrm{s}$).

This affects the input currents as follows. Without virtual input spikes, the input current of neuron i in the first layer would be given by

$$I_i^{(1)}(t) = I_0 + \tau \sum_{j=1}^{N_0} w_{ij}^{(1)} \sum_{t_j} \delta(t - t_j), \tag{19}$$

where N_0 is the number of input neurons and $w_{ij}^{(1)}$ are the input weights. We then approximate all spike times within bin k by t_k^{virt}, which yields

$$I_i^{(1)}(t) \approx I_0 + \tau \sum_{j=1}^{N_0} w_{ij}^{(1)} \sum_{k=1}^{N_\mathrm{T}} n_{jk}\, \delta(t - t_k^{\text{virt}}), \tag{20}$$

where n_{jk} denotes the number of spikes of input neuron j in bin k. Finally, defining the virtual input weights as

$$w_{ik}^{\text{virt}} = \sum_{j=1}^{N_0} w_{ij}^{(1)} n_{jk}, \tag{21}$$

the input current becomes

$$I_i^{(1)}(t) \approx I_0 + \tau \sum_{k=1}^{N_\mathrm{T}} w_{ik}^{\text{virt}} \delta(t - t_k^{\text{virt}}). \tag{22}$$

This formulation is equivalent to replacing the original input layer by N_T virtual neurons, each emitting a single spike at time t_k^{virt}. The virtual weights are not trainable parameters, but are fully determined by the learnable input weights $w_{ij}^{(1)}$ and the spike counts n_{jk}.

While using virtual input spikes discards some temporal information, we find that increasing the temporal resolution beyond 64 bins does not improve classification performance. We thus expect that the use of higher temporal resolutions does not have a relevant impact on the results of this paper.

B.2 Spatial binning

In addition to the temporal binning, we also bin the input data along the spatial direction, which is given by the input neuron indices $j = 1, ..., N_0$. Compared to the time direction, the spatial direction is already discretized into $N_0 = 700$ bins in the SHD dataset, if we regard each neuron as a single bin. To improve the ability of our model to generalize to new data and to reduce the number of learnable parameters, we reduce the spatial resolution. To do this, we squash together multiple input neurons into one by superposing their spike trains, as was done in Cramer et al. (2022). If M_S is the number of neurons that are combined, i.e., the number of neurons in each spatial bin, the number of spatial bins is $N_\mathrm{S} = N_0/M_\mathrm{S}$. Assuming that we already applied temporal binning before and the corresponding spike counts are n_{jk}, the spike count in time bin k and spatial bin $m = 1, ..., N_\mathrm{S}$ is

$$\tilde{n}_{mk} = \sum_{j=(m-1)M_\mathrm{S}+1}^{mM_\mathrm{S}} n_{jk},$$

where we sum over all neurons in the m-th bin. The feedforward weight matrix of the first layer reduces from an $N_1 \times N_0$ to an $N_1 \times N_\mathrm{S}$ matrix with elements $\tilde{w}_{im}^{(1)}$. The virtual weights from the previous section then are

$$\tilde{w}_{ik}^{\text{virt}} = \sum_{m=1}^{N_\mathrm{S}} \tilde{w}_{im}^{(1)} \tilde{n}_{mk}.$$

B.3 SPATIAL SHIFT

Another method to improve the generalization performance is to randomly shift the input spikes in the spatial direction, i.e., across neuron indices. This shift is applied globally for each sample, so every spike in a single sample is shifted equally. However, in each epoch, a different shift is applied, such that the model does not always process the same shifted data, which would lead to a bias and not improve generalization at all. Since we precompute the histograms resulting from temporal and spatial binning before the training, we apply the shift to the spatial bin index m instead of the neuron index itself. The global shift was also applied in Nowotny et al. (2025) and in fact lead to better test accuracies. As done there, we draw the shift distance Δm from a uniform distribution and truncate the decimals to get an integer value[1],

$$\Delta m \sim \text{trunc}\left[\mathcal{U}\left(-(\Delta m_{\text{max}}+1), \Delta m_{\text{max}}+1\right)\right].$$

Here, we add one to the maximum shift distance Δm_{max} since otherwise, due to the truncation, the maximum value would effectively never be drawn. The shifted bin is now computed by adding the random shift distance, $m' = m + \Delta m$. At the boundary towards which the bins are shifted, the spikes are dropped. In the histograms resulting from spatial and temporal binning, they do thus not contribute. At the opposite boundary, neurons without any spikes remain, represented by zeros in the histogram.

B.4 DELAYED COPIES

As done in Nowotny et al. (2025), we augment the input data by adding N_{D} delayed copies of each sample. The k-th copy is delayed by $k \cdot d$, where d is the delay between two copies. Spikes that end up occurring after the trial end T are discarded. As a result, the dataset size increases by a factor of $N_{\text{D}} + 1$.

Using this augmentation, we intend to improve generalization performance by increasing the robustness to temporal shifts in the input data. For spoken digits, translations in time do not change the meaning behind them. Using delayed input copies, we can train the model on the same original sample, but at a different time during the trial.

C HYPERPARAMETER SEARCH

First, we perform a rough hyperparameter search, varying the weight scale w_0, the learning rate α, the logit scale β, the membrane time constant τ, and the constant input drive I_0 over a wide range of values. For each hyperparameter configuration, we simulate 100 epochs with three different random seeds and determine the best test accuracy the model achieves during training in each run. In Figs. 6 and 7, we show these best test accuracies for networks of QIF and LIF neurons, respectively. It should be noted that using the test dataset for determining the best hyperparameter configuration is bad practice since this might cause overfitting on the test set and an overestimation of the accuracy on new data. In this work, we still use the test set for model selection for a fair comparison with other works, which do the same (e.g., see Cramer et al. (2022); Bittar & Garner (2022); Schöne et al. (2024)). A better approach would be to perform cross-validation, as was done, e.g., in Nowotny et al. (2025). However, this would introduce additional computational and conceptual complexity, which we avoid here due to our focus on the detailed analysis of the models themselves.

Next, we focus on the well-performing regions in a finer hyperparameter search, covering smaller volumes in the hyperparameter space in more detail and training for 200 instead of only 100 epochs. For the QIF model, we fix $w_0 = 0.01$ and scan a portion of the region with high accuracies in Fig. 6 with a finer resolution. The resulting test accuracies are shown in Fig. 8. The highest test accuracy we find is $90.8(3)\,\%$ at $\alpha = 3 \cdot 10^{-5}$, $\beta = 300$, $\tau = 0.04\,\text{s}$, and $I_0 = 0.252$.

For the LIF neuron, we fix $w_0 = 10^{-4}$ and perform two separate fine hyperparameter searches, one for low I_0 and τ around $0.2\,\text{s}$, and one for high I_0 and higher τ, since those are the regions with high test accuracies in Fig. 7. In both searches, we extend the search space beyond the boundary of the rough search. The resulting test accuracies are shown in Figs. 9 and 10. We see that the region with

[1]While in Nowotny et al. (2025) it is stated that values are rounded to the nearest integer, the implementation truncates the decimals instead (i.e., rounds toward zero). We therefore also truncate them.

lower I_0 and lower τ performs significantly better than the other region, achieving up to $78.3(16)\,\%$ test accuracy. Table 1 shows the final hyperparameters and metrics for the best-performing QIF and LIF networks as well as the asynchronous LIF network, introduced in Sec. 3.1.

	QIF (best)	LIF (best)	LIF (async)
Initial weight scale w_0	10^{-2}	10^{-4}	10^{-4}
Learning rate α	$3 \cdot 10^{-5}$	$3 \cdot 10^{-7}$	$3 \cdot 10^{-7}$
Logit scale β	300	1,000	30
Membrane time constant τ	20 ms	200 ms	200 ms
Constant input drive I_0	0.252	1.0003	1.00003
Layer sizes	(128, 20)		
Trial duration T	1.4 s		
Batch size N_B	1,000		
Learning rate exp. decay constant τ_α	100 epochs		
Number of spatial bins N_S	35 ($\rightarrow$ 20 neurons per bin)		
Maximum spatial shift Δm_{max}	2 bins ($\hat{=}$ 40 neurons)		
Number of temporal bins N_T	64 ($\rightarrow$ bin size 21.875 ms)		
Number of delayed input copies N_D	2		
Delay per input copy d	0.1 s		
Distinctness (final, test set)	$28.3(14)$ ms	$3.12(9)$ ms	$29.2(46)$ ms
Training accuracy (final)	$97.0(5)\,\%$	$83.5(11)\,\%$	$45.6(18)\,\%$
Test accuracy (final)	$90.1(9)\,\%$	$79.2(13)\,\%$	$50.8(40)\,\%$

Table 1: Hyperparameters of the three models selected for further analysis and the corresponding metrics after training for 500 epochs. For the final accuracy and distinctness values, we report mean and standard deviation across 10 runs with different random seeds.

D LARGE NETWORK RESULTS

To achieve higher accuracies than in Table 1, we use larger networks and perform additional exploratory experiments, also varying hyperparameters that were not part of the hyperparameter searches. Table 2 shows the hyperparameters and metrics of the best large models, both trained for 500 epochs and 10 different random seeds. Especially the LIF network profits from the increased number of neurons. However, we did not find that multiple hidden layers improve the LIF performance. For QIF networks, we found the best accuracy using two hidden layers. Note that in intermediate epochs, the same QIF and LIF networks achieved slightly higher test accuracies of $93.7(6)\,\%$ and $88.0(9)\,\%$, respectively, averaged over 10 seeds. The QIF model achieved this best test accuracy in epoch $172(70)$, the LIF network in epoch $385(79)$. This indicates that early stopping or reducing overtraining might be beneficial, but we did not look into this in more detail since accuracy optimization is not the main focus of this work.

	QIF (large)	LIF (large)
Layer sizes	(512, 512, 20)	(1024, 20)
Logit scale β	100	1,000
Batch size N_B	500	500
Distinctness (final, test set)	$20.6(6)$ ms	$5.64(19)$ ms
Training accuracy (final)	$99.974(13)\,\%$	$94.02(42)\,\%$
Test accuracy (final)	$91.3(8)\,\%$	$85.1(12)\,\%$

Table 2: Hyperparameters and metrics of the two best-effort large QIF and LIF networks, if different from QIF (best) and LIF (best) in Table 1. Metrics are averaged over 10 runs with different random seeds and shown with their standard deviation.

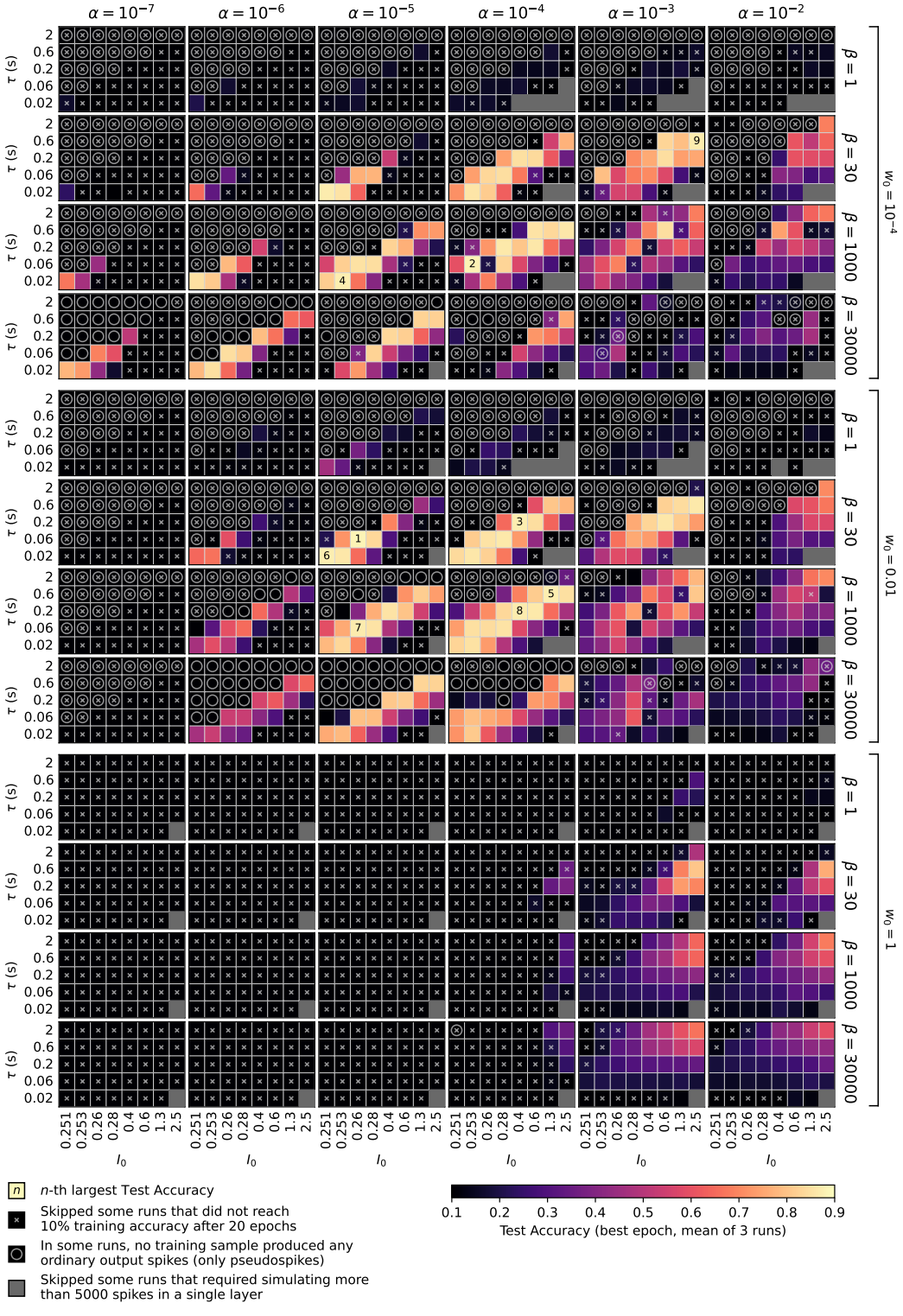

Figure 6: Results of rough hyperparameter search with QIF neurons. For each configuration, we show the best test accuracy achieved across all 100 epochs, averaged over 3 random seeds.

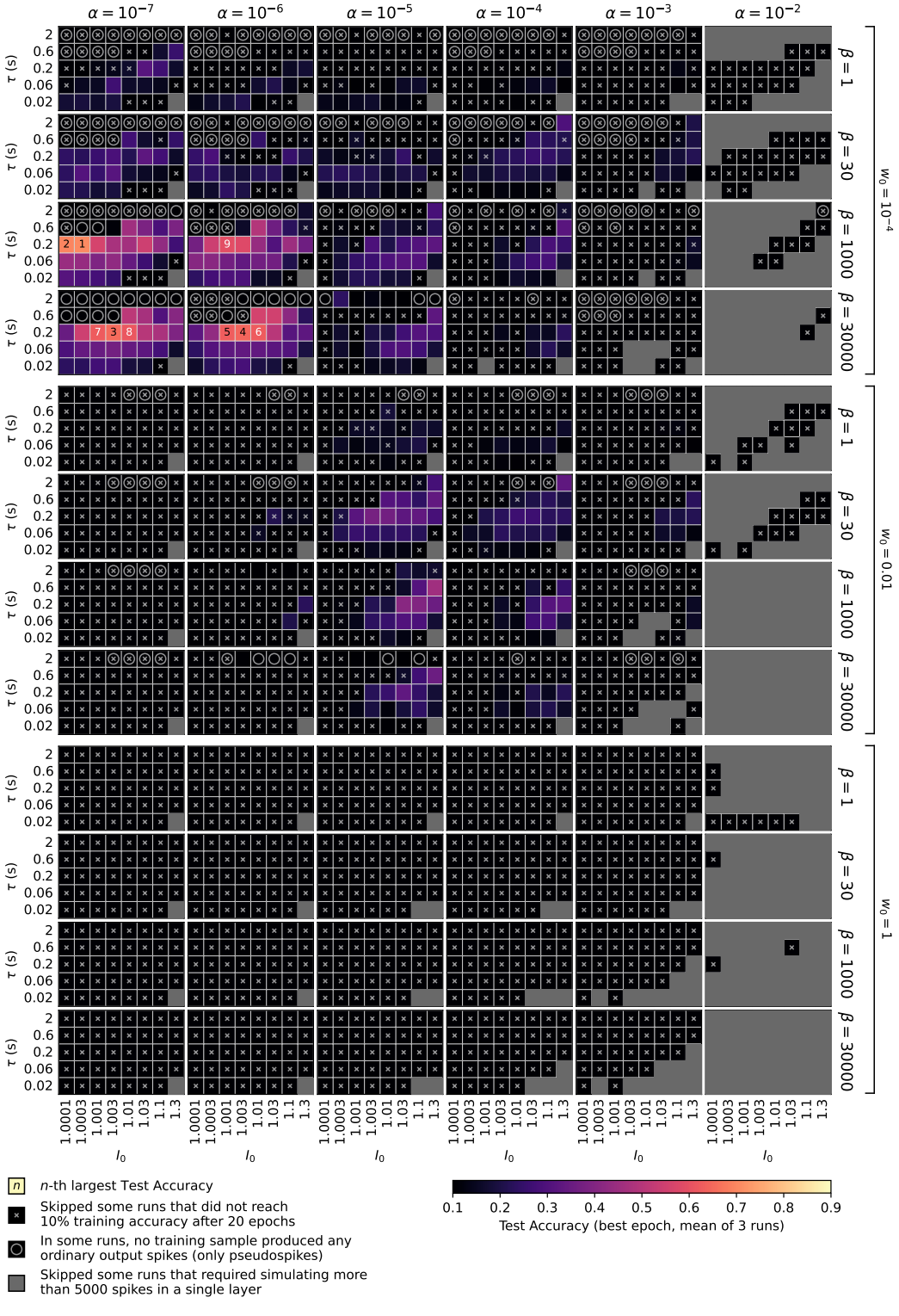

Figure 7: Results of rough hyperparameter search with LIF neurons. For each configuration, we show the best test accuracy achieved across all 100 epochs, averaged over 3 random seeds.

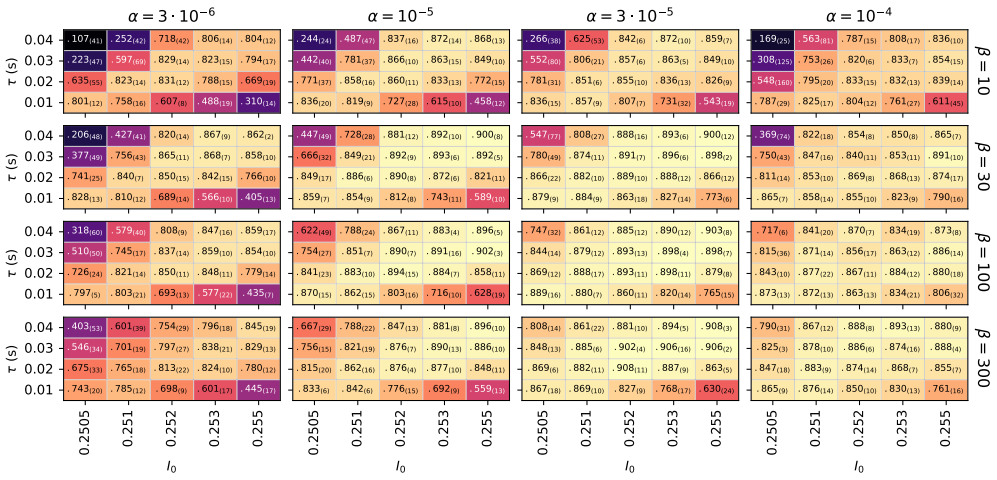

Figure 8: Results of fine hyperparameter search with QIF neurons. For each configuration, we show the best accuracy achieved across all 200 epochs, averaged over 5 random seeds.

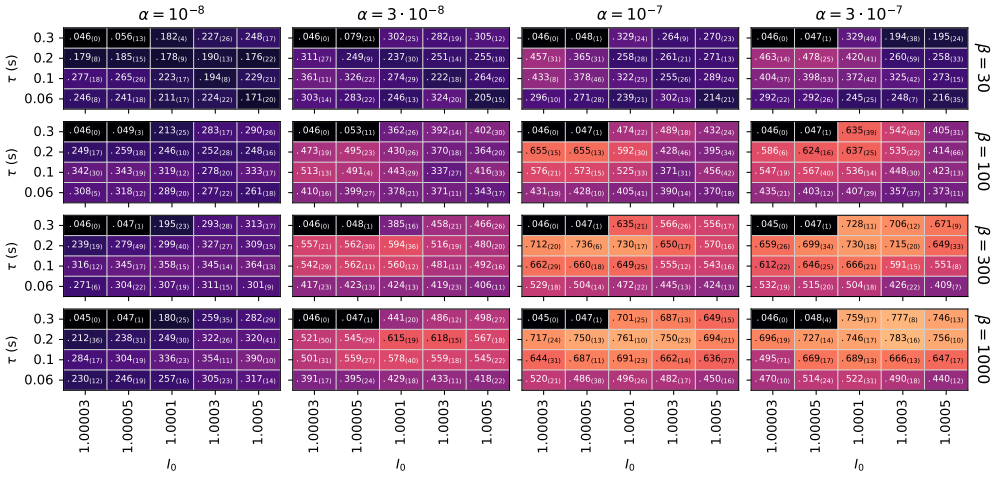

Figure 9: Results of fine hyperparameter search with LIF neurons and low I_0. For each configuration, we show the best accuracy achieved across all 200 epochs, averaged over 5 random seeds.

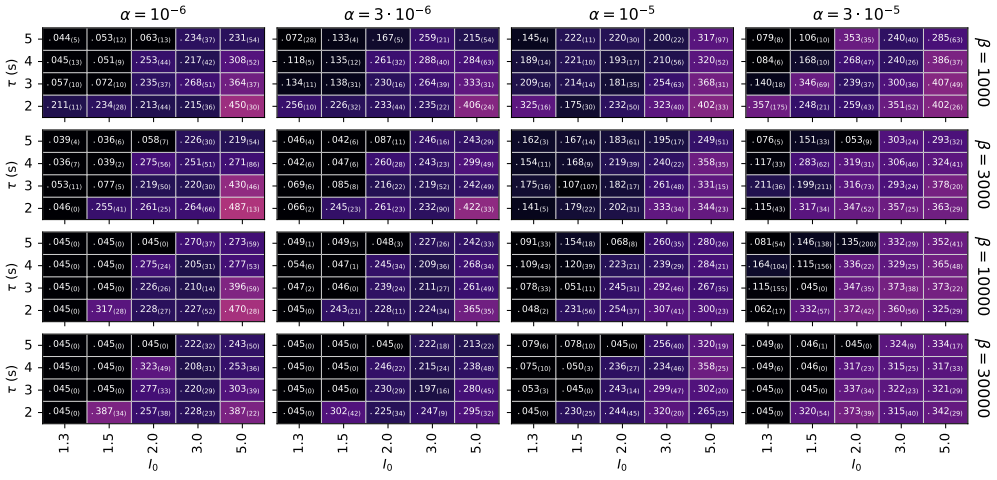

Figure 10: Results of fine hyperparameter search with LIF neurons and high I_0. For each configuration, we show the best accuracy achieved across all 200 epochs, averaged over 5 random seeds.

E ACCURACY-DISTINCTNESS RELATION

In Fig. 2E, we show how the distinctness compares to the final test accuracy for the configurations of the rough hyperparameter search. Here, we examine the accuracy-distinctness relation for both hyperparameter searches in more detail. The results are shown in Fig. 11. We notice that the distinctness is highly dependent on the logit scale β, which is indicated by the color of the points. Configurations with larger logit scales usually have smaller distinctness values. This is likely due to the form of the sample loss function in Eq. (14): Larger logit scales require smaller logit changes to achieve the same change in loss. Thus, the output spikes are not spread as much in time during training, reducing the distinctness. Comparing the QIF and LIF plots, it becomes apparent that QIF neurons allow much larger distinctness values without sacrificing accuracy while LIF neurons suffer from an accuracy reduction when requiring a higher distinctness. However, since hyperparameter grid searches are necessarily restricted to finite regions of the parameter space, a complete characterization of the underlying landscape cannot be guaranteed.

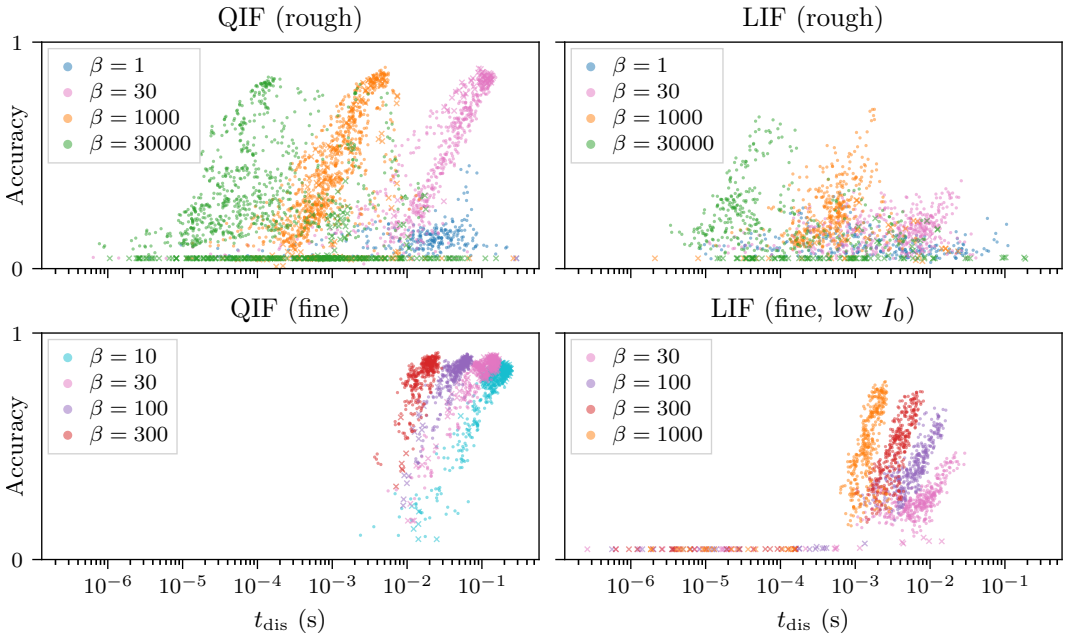

Figure 11: Final test accuracy plotted against distinctness for hyperparameter configurations on the rough and fine search grid, grouped by the neuron type and colored by the logit scale β. To reduce clutter, we exclude the fine LIF grid search with high I_0. Runs with only pseudospikes in the output layer are indicated by crosses instead of dots.

F LANDSCAPE CALCULATION AND ROUGHNESS QUANTIFICATION

F.1 LANDSCAPE CALCULATION

Here, we give a more detailed description of the landscape construction, introduced in Sec. 3.2. Our goal is to evaluate a function $f(\theta)$, e.g. the loss $L(\theta)$, on a two-dimensional subspace $\mathbb{R}^2$ of the parameter space $\mathbb{R}^N$, where N is the number of trainable parameters. We call the subspace the projection plane. The plane is spanned by two normalized and orthogonal direction vectors $u, v \in \mathbb{R}^N$. We choose the primary direction vector u such that it points from the initial model parameters $\theta_0 \in \mathbb{R}^N$ to the final parameters $\theta_K \in \mathbb{R}^N$ after K training steps,

$$u = \frac{\theta_K - \theta_0}{|\theta_K - \theta_0|}. \tag{23}$$

Let $\theta_k \in \mathbb{R}^N$ be the parameter vector after $k = 0, 1, ..., K$ training steps. The secondary direction vector v should point in the direction orthogonal to u along which the training trajectory exhibits its

largest variance. To quantify this, we consider the perpendicular distance vectors $d_k \in \mathbb{R}^N$ from the line through θ_0 and θ_K to the parameter vectors θ_k, which are given by

$$d_k = \theta_k - \theta_0 - ((\theta_k - \theta_0) \cdot u)u. \tag{24}$$

Now, we define the secondary direction vector v as the first principle component (PC1) of the distance vectors d_k for $k = 1, ..., K-1$, i.e., the direction of maximum variance. Here, we exclude d_0 and d_K because they always vanish. As intended, the resulting direction vectors u, v form an orthonormal basis of the projection plane since they satisfy $|u| = |v| = 1$ and $u \cdot v = 0$. At this point, the sign of v is arbitrary. To create a consistent landscape orientation, we choose the sign such that v points in the direction of the first updated parameters θ_1 by requiring $d_1 \cdot v > 0$. We ignore the edge-case $d_1 \cdot v = 0$ since it is highly unlikely.

Next, we construct a coordinate system on the projection plane. Let $\alpha, \beta \in \mathbb{R}$ be the coordinates in the primary (u) and secondary (v) direction, respectively. Both the initial and final parameters should lie on the projection plane. We position the initial parameters θ_0 at the origin of the coordinate system, i.e., at $(\alpha, \beta) = (0, 0)$, and the final parameters θ_K at $(\alpha, \beta) = (1, 0)$. Then, the parameters at (α, β) are given by

$$\theta(\alpha, \beta) = \theta_0 + \alpha s u + \beta s v, \tag{25}$$

where $s = |\theta_K - \theta_0|$ determines the scaling. When projecting a parameter vector $\theta \in \mathbb{R}^N$ onto the plane, the resulting coordinates are

$$\alpha(\theta) = \frac{1}{s}(\theta - \theta_0) \cdot u, \quad \beta(\theta) = \frac{1}{s}(\theta - \theta_0) \cdot v. \tag{26}$$

F.2 Roughness quantification

Here, we describe the landscape roughness quantification in more detail. For the quantification, we focus on the loss and gradient variability on small scales and try to eliminate the effect of large scale variability, such as a desirable general trend towards a loss minimum, as much as possible.

For the loss, we first convolve the loss landscapes with a Gaussian kernel with a given scale $\sigma = 0.03$ (1 would equal the distance between initial and final parameters). Then we subtract the results from the original loss landscapes. This implements a high-pass filter of the loss. Next, we compute the two-point correlation coefficients and bin them according to their distances. Finally, we compute the correlation length as the distance until the correlation coefficients decay to 1/e. Small correlation lengths indicate locally rougher or noisier landscapes. We note that the results of this procedure are qualitatively independent to the precise value chosen for σ (Fig. 12).

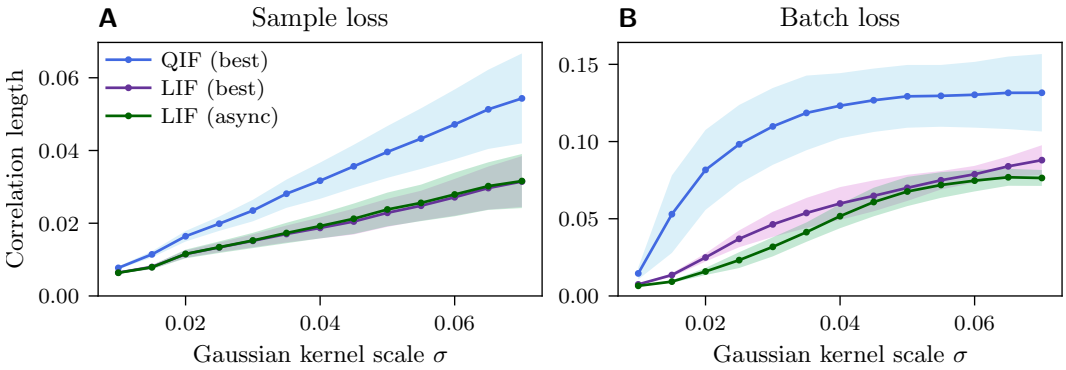

Figure 12: Dependence of sample and batch loss landscape correlation lengths on the Gaussian kernel size σ. Lines and points are means over 5 seeds, filled areas represent standard deviations. The QIF model has a larger correlation length than both LIF models, independent of the specific σ.

For the gradient, we use a different type of analysis as the gradients decorrelate on scales similar to the grid spacing (0.005) for LIF neurons. Specifically, we determine the cosine similarity of the gradients of neighboring grid points, averaged over all neighbor pairs in both the α and β directions.

High average cosine similarities (close to 1) indicate high local spatial correlations and thus a small amount of noise. Small average cosine similarities (close to 0) indicate that the neighboring angles are on average 90° apart, as would be the case for a uniform random angle distribution.

G Example of an LIF gradient discontinuity

In Fig. 13 we show the LIF voltage dynamics around the gradient discontinuity at arrow b in Fig. 4.

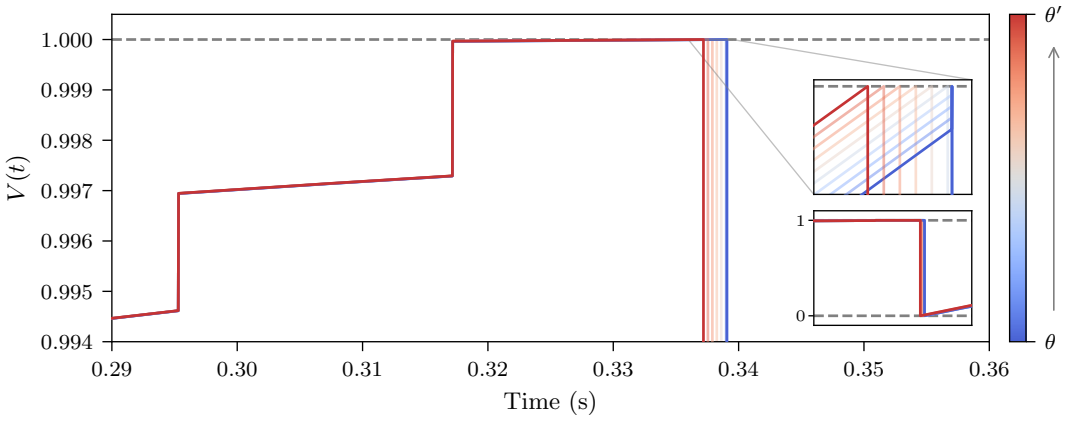

Figure 13: Change of the voltage dynamics of the hidden neuron that results in the gradient discontinuity along arrow b in Fig. 4. Coloring as in Fig. 1F. See description of Fig. 4 in the main text for details.

H Example of a sharp QIF loss transition

Here, we present an example of a loss jump in a QIF network. In Fig. 14A, we show the loss landscape of a single sample containing the full training trajectory (200 epochs). Zooming in by a factor of 20, in Fig. 14B, we notice that there seems to be a loss jump around the initial parameters (indicated by a triangle). Zooming in even further, by a factor of 200, in Fig. 14C, we see that the jump is a sharp but continuous transition. To make this even clearer, we show the loss along the training trajectory in this region as a line plot in Fig. 14F. There, we see a very steep decline, but not a discontinuity. On the other hand, the loss gradient angle seems to exhibit a discontinuous transition in Fig. 14D and G. The fact that the loss changes very rapidly in this region is also reflected by a very large loss gradient amplitude, shown in Fig. 14E and H. However, when zooming in even further (Fig. 14I, J and K), this gradient jump also vanishes and turns out to be a continuous but sharp transition. The accompanying abrupt but continuous change of a spike time in the network is shown in Fig. 1E.

I Example of a QIF gradient discontinuity

In contrast to the example shown in Sec. H and Fig. 14, where both the loss and its gradient seemed discontinuous from a large-scale point of view but turned out to be continuous at sufficiently small scales, we show an example of a true gradient discontinuity, which does not vanish at small scales, in Fig. 15. In this example, the loss shows no sharp transition and the discontinuities of the gradient angle and amplitude are small compared to their nominal value. Because of this, we do not expect such cases to have a significantly negative effect on training.

J Additional landscapes

In Fig. 3A and B, we only show the loss and gradient angle landscapes of a single sample. For completeness, we also show the gradient amplitude landscape of this sample in Fig. 16A. In Fig. 16B-J, we further show the landscapes of three additional samples. To make individual jumps more

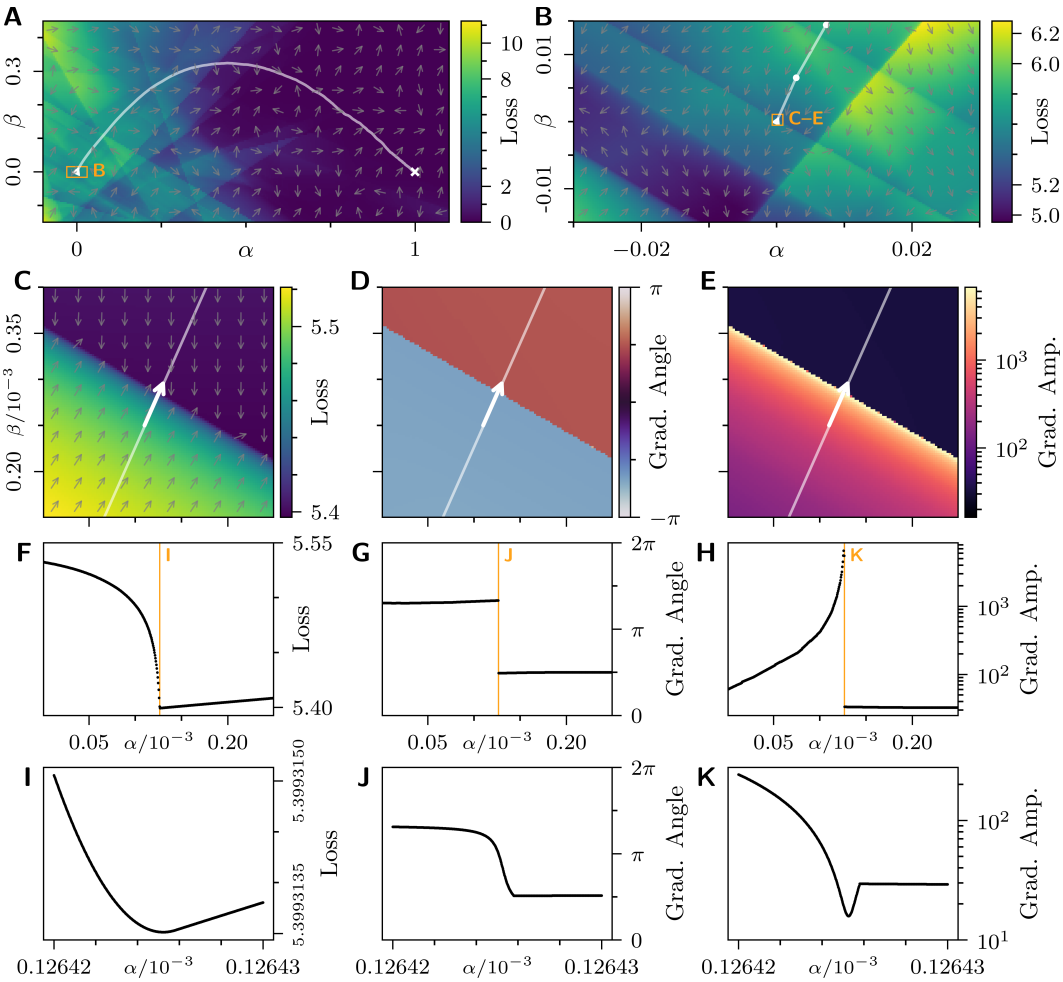

Figure 14: Example region of a QIF sample landscape exhibiting a sharp loss transition. (A) Large-scale view of the loss landscape including the full training trajectory (white line). (B) Loss landscape, zoomed in to the initial parameters, around which the sharp loss transition occurs. (C-E) Loss, gradient angle and gradient amplitude landscapes, zoomed in to the sharp loss transition. (F-H) Corresponding line plots along the training trajectory. At this scale, the loss seems continuous (F) but the gradient does not (G,H). The gradient amplitude shows a very steep increase (H). (I-K) Same as (F-H) but zoomed in further to the sharp transition point, revealing that both the loss and its gradient are actually continuous.

obvious, we zoom in by a factor of 20 in Fig. 17. In Fig. 18 we show batch loss and gradient angle landscapes for four additional random seeds, to demonstrate that the qualitative appearance and the features described in Sec. 3.2 do not depend on the seed.

K Loss changes along training path

In Fig. 5A, we give some examples of how the loss changes between two training steps. Some of the loss curves vary erratically, especially for LIF neurons. In some cases, the loss even increases. Here, we measure the loss difference $\Delta L = L(\theta_{k+1}) - L(\theta_k)$ for all training steps in 200 epochs (1,800 steps in total).

If the loss landscape is locally well approximated by a linear expansion, the loss function satisfies

$$L(\theta_k + \Delta\theta_k) \approx L(\theta_k) + \Delta\theta_k \cdot \nabla L(\theta_k) \tag{27}$$

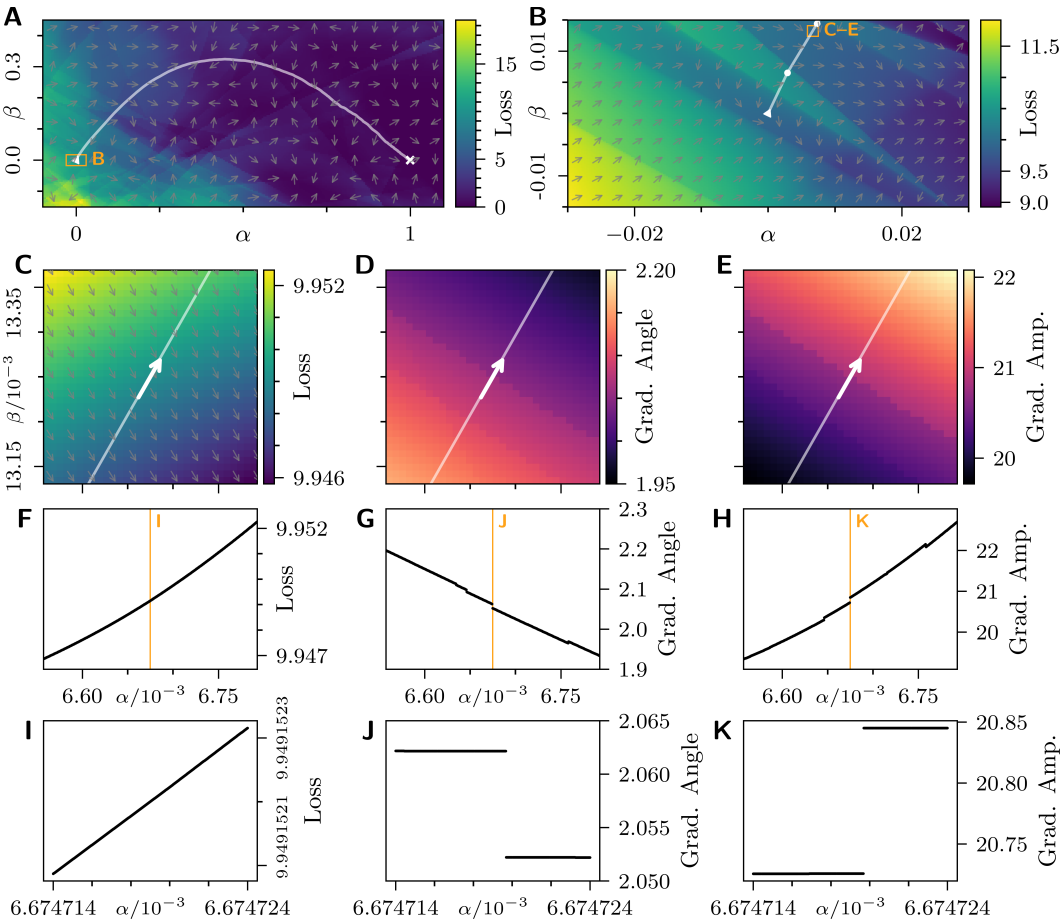

Figure 15: Example region of a QIF sample landscape exhibiting a true loss gradient discontinuity. Analogous to Fig. 14. (A) Large-scale view of the loss landscape including the full training trajectory (white line). (B) Loss landscape, zoomed in to the initial parameters. (C-E) Loss, gradient angle and gradient amplitude landscapes, zoomed in to a region with gradient discontinuities. (F-H) Corresponding line plots along the training trajectory. (I-K) Same as (F-H) but zoomed in further to a single gradient discontinuity, showing that it does not vanish, as opposed to Fig. 14I-K.

for sufficiently small updates $\Delta\theta_k = \theta_{k+1} - \theta_k$. Based on this first-order approximation, we define the expected loss difference as

$$\Delta L_{\text{exp}}(\theta_k) = \Delta\theta_k \cdot \nabla L(\theta_k). \tag{28}$$

Comparing the actual and expected loss differences therefore provides a measure of how accurately the first-order approximation describes the loss along the training trajectory (Fig. 19). Note that there are steps in all models with a positive expected loss difference, which result from using the Adam optimizer, which does not strictly follow the direction of steepest descent.

Gradient descent relies on an (at least roughly) accurate first-order approximation. As a quantitative measure of this, we determine the fraction of training steps that reduce the loss ($\Delta L < 0$), when a reduction in loss is expected ($\Delta L_{\text{exp}} < 0$). This corresponds to the number of points in the lower left quadrant of Fig. 19 divided by the number of points in the left half. For the best-performing QIF model, $90.0(11)\,\%$ of steps with an expected loss reduction also reduce the loss, while for the best-performing and asynchronous LIF models the fractions are only $73.4(12)\,\%$ and $61.3(19)\,\%$, respectively (mean and std. of 10 seeds, each trained for 200 epochs, corresponding to 1,800 update steps).

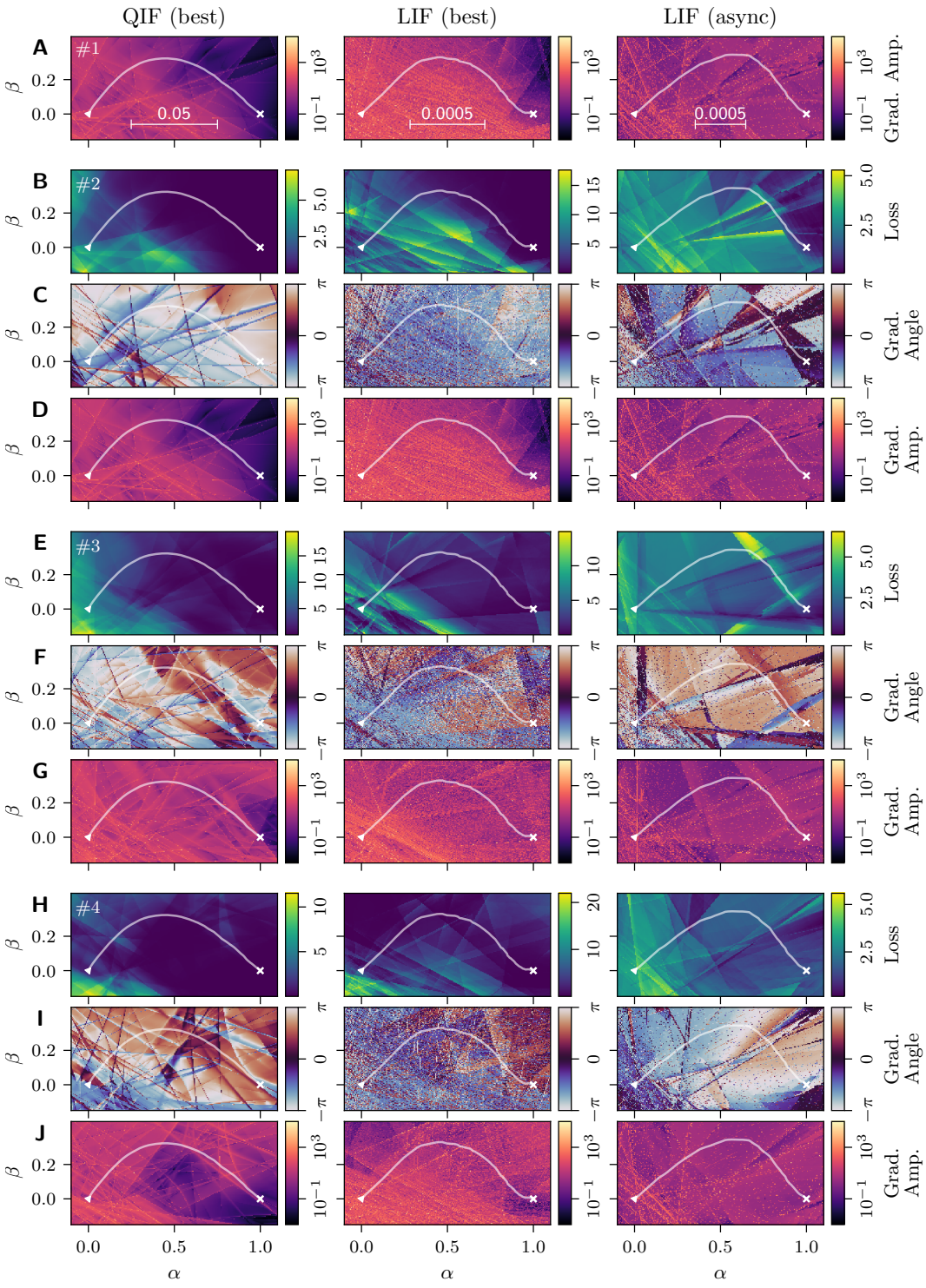

Figure 16: Additional landscapes. (A) Gradient amplitude landscapes of the same sample shown in Fig. 3A and B. (B-J) Loss, gradient angle and gradient amplitude landscapes for three additional samples.

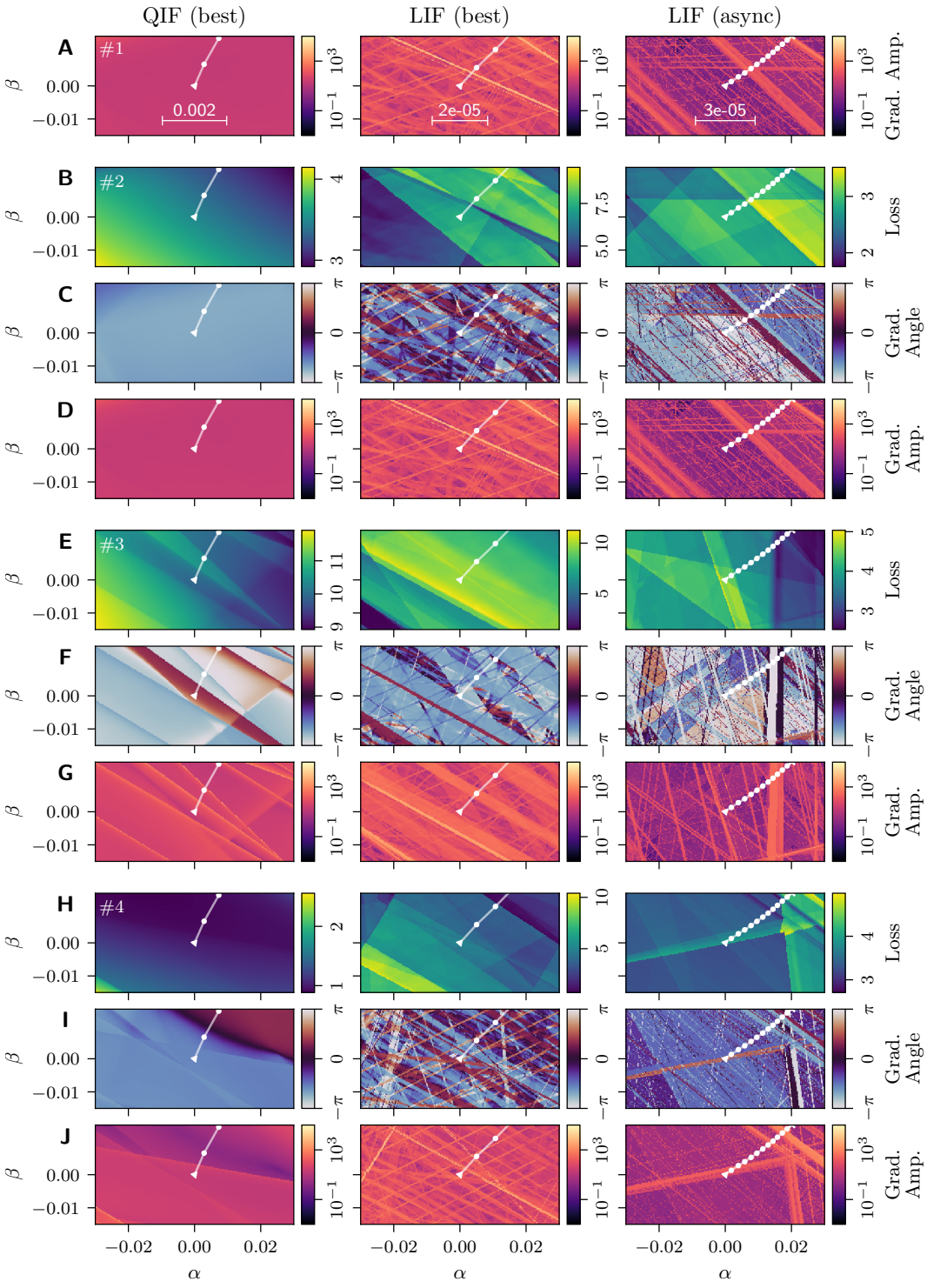

Figure 17: Same landscapes as in Fig. 16 but zoomed in by a factor of 20. Individual training steps are indicated by white circles.

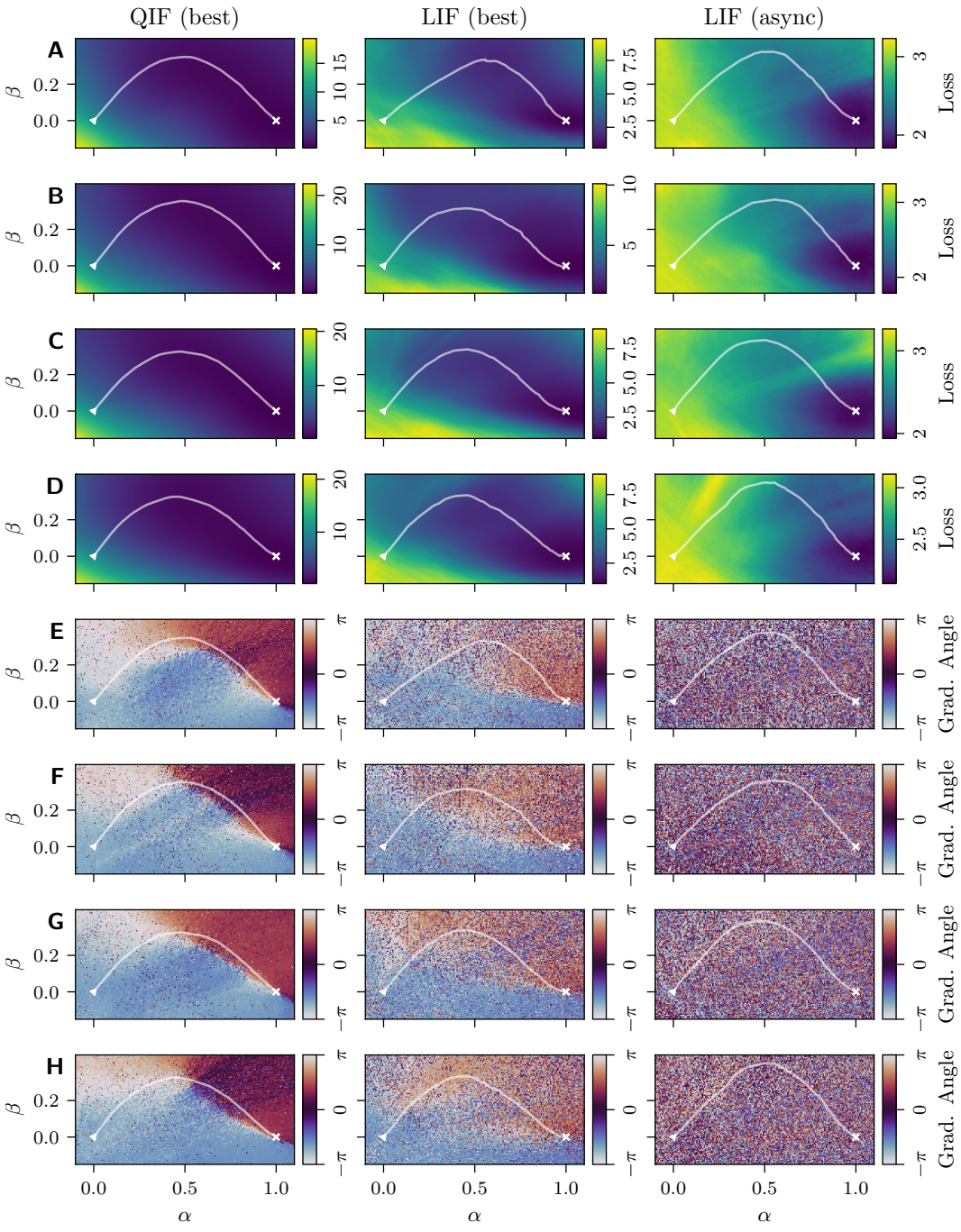

Figure 18: Batch loss landscapes (A-D) and batch gradient angle landscapes (E-G) for 4 additional seeds. The qualitative appearance of the landscapes does not change significantly for different seeds.

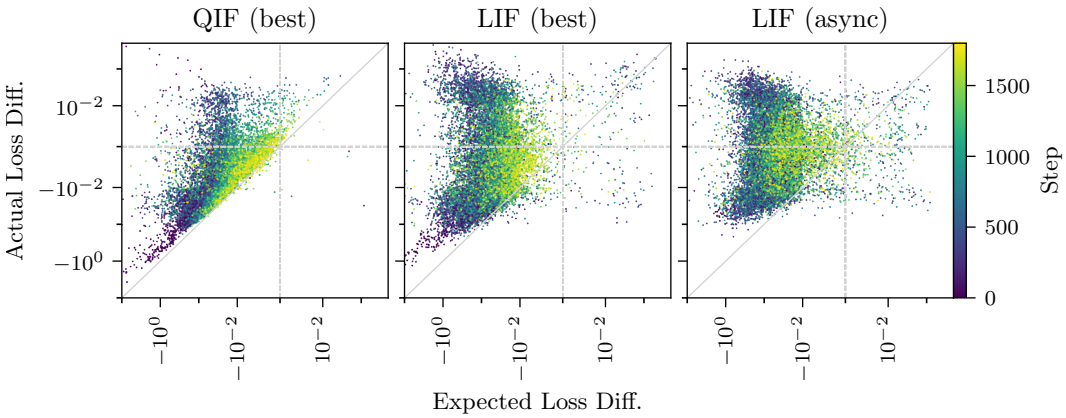

Figure 19: Actual loss differences plotted against expected ones for all training steps. Corresponding step numbers are indicated by different colors. Each point represents a single update step of 10 training runs with different seeds (1,800 steps each).

L Implementation & Computations

Most computations required for this work were implemented using the JAX Python library (Bradbury et al., 2018), which enables just-in-time compilation and automatic differentiation, and executed on GPU nodes (Nvidia A40, 48GB) on an HPC cluster. For training models, i.e., the results given in Sec. 3.1, we usually used a single GPU. Each epoch has a runtime of around 2 to 3 seconds. The total training time for 500 epochs (including data loading and compilation) is typically between 20 and 40 minutes, depending mainly on how many spikes need to be simulated for the specific model. For landscape generation (Figs. 1, 3 and 4) and probing of intermediate points along the training path (Figs. 4, 5 and 14), we usually used four GPUs in parallel. These tasks may run for multiple hours, depending on the resolution of the grid. By far the most costly computations were the rough and fine hyperparameter searches since they require 17,280 and 4,800 model training runs, respectively. The total runtime of both searches combined is about 70 days (using a single GPU per training run). However, using parallel jobs, this runtime can be reduced significantly. We did not measure detailed memory usage statistics. However, by splitting parallelized tasks into batches, insufficient GPU memory can be easily avoided, at the cost of longer runtimes. For example, when computing the loss and gradient landscapes, we evaluated batches of 20 points in parameter space to avoid GPU memory issues. Note that performance optimization was not a major focus in this work.

In addition to JAX, we use Optax (DeepMind et al., 2020) for optimization, Chex (DeepMind et al., 2020) for testing utilities, scikit-learn (Pedregosa et al., 2011) for principle component analysis, pandas (McKinney, 2010; The pandas development team, 2020) for data management, and Matplotlib (Hunter, 2007) for visualizations.