How to realize efficient Spiking Neural Networks?

Adalbert Fono[1,4], **Holger Boche**[2], **Gitta Kutyniok**[1,3,4,5]

[1]Ludwig-Maximilians-Universität München [2]Technical University of Munich [3]University of Tromsø
[4]Munich Center for Machine Learning (MCML) [5]DLR-German Aerospace Center

Abstract

Spiking neural networks (SNNs) have been proposed as an (energy-)efficient alternative to conventional artificial neural networks. However, the aspired benefits have not yet been realized in practice. To gain a better understanding of why this gap persists, we theoretically study both discrete-time and continuous-time models of leaky integrate-and-fire neurons. In the discrete-time model, which is a widely used framework due to its amenability to conventional deep learning software and hardware approaches, we analyze the impact of explicit recurrent connections on the network size required to approximate continuously differentiable functions. We contrast this view by investigating the computational efficiency of digital systems that simulate spike-based computations in the continuous-time model. It turns out that even in well-behaved settings, the computational complexity of this task may grow super-polynomially in the prescribed accuracy. Thereby, we exemplarily highlight the intricacies of realizing potential strengths in the biological context, namely recurrent connections and computational efficiency, of spike-based computations on digital systems.

1 Introduction

Artificial neural networks (ANNs) have achieved success in a wide range of applications. However, implementations of (large-scale) ANNs often require the allocation of considerable computational resources, which are projected to increase on the current digital computing platforms (Thompson et al. 2021). Spiking neural networks (SNNs) are envisioned to be more closely aligned with the structure of computations in biological neural networks to exploit their intrinsic benefits, foremost their energy efficiency (Maass 1997; Mehonic et al. 2024; Rathi et al. 2023).

The adoption of spikes—asynchronous, point-like electrical pulses in the biological context—as the means of communication between neurons represents the key innovation of SNNs (Gerstner et al. 2014). This event-driven nature distinguishes them from conventional ANNs, which rely on synchronous information propagation, suited to digital computing architectures enabling efficient parallelization of computations, thereby accelerating processing in ANNs (Pandey et al. 2022; Silvano et al. 2023). In contrast, SNNs, by design, do not naturally conform to this parallelization paradigm; consequently, specialized hardware platforms (neuromorphic hardware) that align with their intrinsic properties are being developed.

The term neuromorphic originally described systems emulating specific aspects of biological neural networks. Now, it more broadly refers to systems that manifest brain-inspired properties, including fine-grained parallelism, reduced precision computing, and in-memory computing (Mehonic et al. 2024; Christensen et al. 2022). This resembles biological systems, in which software and hardware are intertwined: the abstract computational model is inherently embedded in the biological environment. In contrast, to leverage established methods used to train and operate ANNs, e.g., stochastic gradient descent with backpropagation and their optimized software implementations, one objective has been to adapt SNNs to the structure of these algorithms and to the architecture of (commercially) available digital neuromorphic hardware (Orchard et al. 2021; Gonzalez et al. 2023).

A major step therein has been the reformulation of spiking neuron behavior from continuous-time dynamics governed by differential equations into sequentially and synchronously operating discrete-time models. This naturally raises the question of how capable the resulting discrete-time models are, i.e., to what extent they retain the advantages of their biological counterparts. Indeed, the discretization and subsequent digital implementation of SNNs seem to introduce degradations compared to the biological paradigm. Notably, embedding SNNs into conventional deep learning pipelines on digital hardware diminishes their potential energy efficiency, even when executed on digital neuromorphic platforms (Dampfhoffer et al. 2024).

Is this potential degradation due to insufficient tools or inevitable in the alignment process of the model to the hardware? We tackle this question from two angles and highlight in both cases the intricacies when converting SNNs into discrete-time, digital models. First, we study the impact of recurrent connections, a key feature in the non-trivial connectivity structure of biological networks (Gerstner et al. 2014; Vidal-Saez, Vilarroya, and Garcia-Ojalvo 2024), in discrete-time SNNs. Second, we examine the algorithmic complexity of SNN implementations on digital hardware.

Contributions We analyze the expressive power of SNNs with discrete-time dynamics, based on the leaky integrate-and-fire (LIF) neuron model. Here, expressivity refers to the capacity and efficiency of a model class to represent or approximate a given class of functions. We establish approximation rates for continuous functions under suitable regularity conditions by exploiting explicit recurrent connections. Although this provides further evidence that discrete-time models can preserve the expressive power of their more general continuous-time variants, the consideration of recurrent connections does not generally benefit the model in our construction.

We offer a complementary perspective by analyzing the computational efficiency of digital systems that emulate/simulate spike-based computations based on the continuous-time LIF model. Specifically, we study the computational complexity of approximating the solutions to the underlying differential equations on digital hardware modeled by Turing machines (Turing 1936). Our main insight is that even low-complexity input signals can produce high-complexity solutions in terms of computation steps, thereby complicating the (energy-)efficient realization of brain-like computations on digital hardware.

2 Methods

Continuous-time Model We consider a simple model class of neuronal dynamics still retaining biological plausibility, the *Integrate-and-Fire* (IF) models (Gerstner et al. 2014). They generally consist of two components: (i) the time evolution of the *potential* $u(t)$ of a neuron given a *current* $I(t)$ and (ii) the spike generation mechanism via thresholding operations. In the *Leaky IF* (LIF) model, a time-continuous dynamical system describes the evolution of the potential via a system of linear differential equations,

$$\begin{cases} \tau_m \frac{\mathrm{d}u}{\mathrm{d}t}(t) = -(u(t) - u_{\text{rest}}) + I(t), \\ \tau_s \frac{\mathrm{d}I}{\mathrm{d}t}(t) = -I(t) + I_e(t), \end{cases} \tag{1}$$

where $\tau_m, \tau_s > 0$ are the *membrane* and *synaptic time constant*, respectively, and $I_e(t)$ denotes an *external input current*. When u reaches a certain value, the *firing threshold* $\vartheta > 0$, the neuron emits a spike and subsequently u is reset to the *resting potential* u_{rest} while the spike triggers postsynaptic currents in downstream neurons. The temporal dynamics in a layered network of spiking neurons are then typically (mathematically) abstracted by conceiving spikes as point processes localized in time expressed as a sum of Dirac delta distributions, the so-called *spike train* $S(t) = \sum_s \delta(t-s)$, where s iterates over the spike times of a neuron. Taking into account the network structure and assigning weights to the edges, the so-called *synapses*, yields the simplified dynamic of the current of neuron i in layer ℓ:

$$\frac{\mathrm{d}I_i^\ell}{\mathrm{d}t}(t) = -\frac{1}{\tau_s} I_i^\ell(t) + \sum_j W_{i,j}^\ell S_j^{\ell-1}(t) + \sum_j V_{i,j}^\ell S_j^\ell(t), \tag{2}$$

where W^ℓ is the weight matrix for feedforward connections and V^ℓ is for explicit recurrent connections within each layer. Note that via the potential, neurons propagate their state through time, exhibiting implicit recurrence in time.

Discrete-time Model To render the time-varying dynamics compatible with synchronous neural network computations, the solution of the differential equation is approximated in discrete time, e.g., using Euler methods. Furthermore, the spike train is represented as a binary sequence, allowing the spike generation and reset mechanisms to be incorporated into the framework. This yields the following model, where w.l.o.g. $u_{\text{rest}} = 0$ and $\vartheta = 1$ is assumed.

Definition 1. *A* recurrent discrete-time SNN (RLIF-SNN) $\Phi := ((W^\ell, V^\ell, b^\ell, u^\ell(0))_{\ell \in L}, \alpha, \beta, T)$ *with* $T \in \mathbb{N}$ *time steps (latency),* $L \in \mathbb{N}$ *layers with widths* $(n_0, \dots, n_{L+1}) \in \mathbb{N}^{L+2}$ *(i.e., with input dimension* $n_0 := n_{in}$ *and output dimension* $n_{L+1} := n_{out}$*), weight matrices* $W^\ell \in \mathbb{R}^{n_{\ell-1} \times n_\ell}$ *and* $V^\ell \in \mathbb{R}^{n_\ell \times n_\ell}$*, bias vector* $b^\ell \in \mathbb{R}^{n_\ell}$*, decay parameters* $\alpha, \beta \in [0,1]$*, initial potential* $u^\ell(0) \in \mathbb{R}^{n_\ell}$*, and initial spike activations* $(S^0(t))_{t \in [T]} \in \mathbb{R}^{n_{in} \times T}$ *follows the dynamics*

$$\begin{cases} I^\ell(t) = \alpha I^\ell(t-1) + W^\ell S^{\ell-1}(t) + V^\ell S^\ell(t-1), I^\ell(0) = 0, \\ p^\ell(t) = \beta u^\ell(t-1) + I^\ell(t) + b^\ell, \\ S^\ell(t) = H(p^\ell(t) - \mathbf{1}_{n_\ell}), \quad S^\ell(0) = 0, \\ u^\ell(t) = p^\ell(t) - S^\ell(t), \quad \forall l \in [L], t \in [T], \end{cases}$$

where H *denotes the Heaviside step function.*

To turn an RLIF-SNNs Φ into a viable computation model, a task needs to be encoded in the initial spike activations, and the resulting binary spike activations in the final layer need to be converted into the task domain. This is achieved via coding functions that can be thought of as additional layers in the network. We employ a common choice in our static data setting (Eshraghian et al. 2023), which captures a wide range of possible applications, given by an encoder-decoder pair $E : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{in}} \times T}$ and $D : \{0,1\}^{n_L \times T} \to \mathbb{R}^{n_{\text{out}}}$ parameterized by $a \in \mathbb{R}^T$, $b^{L+1} \in \mathbb{R}^{n_{\text{out}}}$, and $W^{L+1} \in \mathbb{R}^{n_{\text{out}} \times n_L}$:

$$\begin{cases} E(x)(t) = x \quad \forall t \in [T], \\ D\Big(\big(S(t)\big)_{t \in [T]}\Big) = \sum_{t=1}^T a_t (W^{L+1} S(t) + b^{L+1}). \end{cases}$$

Definition 2. *An RLIF-SNN* Φ *with encoder* E *and decoder* D *realizes the mapping* $R(\Phi) : \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_{out}}$ *given by*

$$R(\Phi)(x) = D\Big(\big(S^L(t)\big)_{t \in [T]}\Big) \quad \text{with } S^0(t) = E(x)(t) = x.$$

Complexity Theory We next formalize our notion of computational complexity to analyze the computations underlying LIF neurons. Classical complexity theory examines the processing steps of Turing machines, which serve as an abstract model of digital computations, on discrete sets (Arora and Barak 2009). Extending the natural, discrete domain of digital computers to continuous domains such as the real numbers requires approximating exact real quantities by rational numbers; thus, restricting computations to finitely many discrete values processed sequentially. While discretizing the inputs and parameters of an analog/continuous-time system to obtain a rational representation is typically unproblematic, the resulting solutions are not necessarily rational. Consequently, it is essential to explicitly consider the continuous domain when addressing questions of computational complexity.

To quantify the computational complexity on a continuous domain, the number of iterations to achieve a prescribed approximation accuracy is measured. In this way, the complexity of operations on real numbers can be systematically assessed. Classically, problems of low complexity are those for which computation time grows at most polynomially with respect to the relevant parameters. Polynomial-time problems are generally considered tractable in practice, whereas super-polynomial problems are typically regarded as infeasible. Thus, computational complexity is central in evaluating the practical feasibility of implementing a given problem. Next, we present a formal framework to study the complexity of simulating continuous-time LIF neurons on digital hardware (Ko 1991; Weihrauch 2000).

Definition 3. *A real number $x \in \mathbb{R}$ is* polynomial-time computable *if there exists a Turing machine $M : \mathbb{N} \to \mathbb{Q}$ such that M converges effectively to x, i.e., for all $n \in \mathbb{N}$*

$$|x - M(n)| \leq 2^{-n}, \tag{3}$$

and a polynomial $p \in \mathbb{N}[X]$ such that (3) *holds after at most $p(n)$ computation steps of M.*

To formalize the notion of complexity for functions, we employ oracle Turing machines. These are standard Turing machines M_γ equipped with an additional oracle γ that can return the function value $\gamma(\cdot)$ in a single computational step. This framework enables a clear separation between the computation of a function's input and its value.

Definition 4. *A function $f : [a, b] \to \mathbb{R}$ is* computable *if there exists an Oracle Turing machine $M_\gamma : \mathbb{N} \to \mathbb{Q}$ such that for any oracle $\gamma : \mathbb{N} \to [a, b]$ that effectively converges to $x \in [a, b]$ we have for all $n \in \mathbb{N}$*

$$|f(x) - M_\gamma(n)| \leq 2^{-n}. \tag{4}$$

If there additionally exists a polynomial $p \in \mathbb{N}[X]$ such that (4) *holds after at most $p(n)$ computation steps of M_γ for all $n \in \mathbb{N}$, f is* polynomial-time computable*.*

Remark 5. *A function $f : [a, b] \to \mathbb{R}^d$ is (polynomial-time) computable if its components are (polynomial-time) computable. We write Comp$([a, b]; \mathbb{R}^d)$ and Pol$([a, b]; \mathbb{R}^d)$ for the set of (polynomial-time) computable functions on $[a, b]$.*

Equipped with the notion of polynomial-time computable functions, we can discuss the concept of a complexity blowup. This phenomenon occurs when input signals of low complexity give rise to output signals of high complexity under a given operation. Here, low complexity refers to polynomial-time computability. Thus, if complexity blowup occurs, the resulting output is no longer polynomial-time computable, rendering the practical feasibility of such computations questionable. It is important to note that computation time is measured relative to the desired output accuracy. Hence, even in the presence of a complexity blowup, practical computation may remain feasible if low-accuracy approximations of the output are sufficient.

Definition 6. *An operator $T : \mathcal{F} := \{f \mid f : [a, b] \to \mathbb{R}^d\} \to \mathcal{F}$,* preserves computability *or* polynomial-time computability *if $T\left(\textit{Comp}([a, b]; \mathbb{R}^d)\right) \subseteq \textit{Comp}([a, b]; \mathbb{R}^d)$ or $T\left(\textit{Pol}([a, b]; \mathbb{R}^d)\right) \subseteq \textit{Pol}([a, b]; \mathbb{R}^d)$, respectively. A computability preserving operator T exhibits* complexity blowup *if $T\left(\textit{Pol}([a, b]; \mathbb{R}^d)\right) \nsubseteq \textit{Pol}([a, b]; \mathbb{R}^d)$.*

3 Results

Approximation rates in RLIF-SNN model It is well established that discrete-time LIF networks realize Boolean functions when the coding layers are disregarded. Also considering conding layers, the model becomes equivalent to ANNs with Heaviside activation for $T = 1$, directly inheriting universal approximation properties, e.g., with respect to Boolean and continuous functions (Khalife, Cheng, and Basu 2024). However, a more detailed analysis of the computational structure of SNNs yields deeper insights. Neglecting explicit recurrent connections in the RLIF-SNN model, i.e., by setting $V^\ell = 0$, this model class realizes piecewise constant functions defined over polyhedral regions. Moreover, approximation rates beyond universal approximation results have been established for Lipschitz-continuous (and more generally, uniformly continuous) functions, quantifying the network size required to achieve a certain approximation accuracy, albeit still in the shallow regime ($L = 2$) with low latency ($T = 1$) (Nguyen et al. 2025).

While understanding the effects of depth and higher latency remains an important direction, we extend previous analysis by incorporating explicit recurrent connections. Although our findings do not address the role of depth, we demonstrate that, under certain conditions, the inclusion of recurrent connections can reduce the number of neurons required to achieve a given approximation accuracy.

Theorem 7. *Let $f \in \mathcal{C}^1(C, \mathbb{R}^m)$ be defined on an open hypercube $C \subset \mathbb{R}^n$ such that f has bounded total derivative df, i.e., $M := \sup_{x \in C} \|df(x)\|_2 < \infty$. For all $\varepsilon > 0$, there exists a RLIF-SNN Φ with $L = 2$ and $T = (K(\mu) + 1)T_r(\nu) + 2$ such that*

$$\sup_{x \in C} \|R(\Phi)(x) - f(x)\|_2 \leq \varepsilon,$$

where

$$T_r(\nu) := \max\left\{2, \left\lceil \sqrt{n} \frac{\operatorname{diam}_\infty(C)}{K(\mu)} \frac{M}{\nu} \right\rceil \right\},$$

$$K(\mu) := \min_{\substack{\xi, \theta > 0 \\ \xi\theta = \mu}} \left\{ \left\lceil \sqrt{n} \frac{\operatorname{diam}_\infty(C)}{2 \min(\omega^\dagger(\xi), \theta)} \right\rceil \right\},$$

with $\nu + \mu = \varepsilon$, $\omega^\dagger$ denoting the generalized inverse of a modulus of uniform continuity of df with respect to $\|\cdot\|_2$, and $\operatorname{diam}_\infty(C) := \sup_{x,y \in C} \|x - y\|_\infty$. Moreover, the width parameter $n = (n_1, n_2)$ are given by

$$n_1 = n + 1,$$
$$n_2 = K(\mu)^n (n + 1) + 3.$$

Remark 8. *One can generalize the statement to functions defined on compact sets $\Omega \subset \mathbb{R}^n$ under some technical conditions (in particular, if Ω can be embedded into a hypercube while maintaining the prescribed regularity conditions on a $\delta(\varepsilon)$-tube around Ω), without impacting the order of the approximation rate; see Appendix A.*

Compared to the model without explicit recurrent connections analyzed in Nguyen et al. (2025), the obtained approximation rates are only partially improved. For certain functions, the rates are in fact worse, suggesting that the inclusion of recurrent connections does not always provide a clear

advantage—although this observation may reflect suboptimal aspects of our constructive proof; see Appendix A for details. Conversely, the reduction in network size comes at the cost of increased latency, and consequently, a potentially higher number of spikes. Thus, it remains unclear whether incorporating recurrent edges enhances performance or enables compact energy-efficient networks, warranting further investigation, in particular, on dynamic, event-driven tasks.

Complexity Blowup of LIF neuron We consider the dynamics of a LIF neuron governed by the system of ordinary differential equations (ODEs) described in (1). Our objective is to understand the complexity of emulating these dynamics on digital hardware by examining the potential occurrence of complexity blowup in this context. We say that an ODE exhibits complexity blowup if its solution operator does.

Note that (1) constitutes a system of first-order linear ODEs with constant coefficients, driven by the input signal $I_e(t)$ and producing the output signal $(u(t), I(t))$. Recall that these dynamics describe a LIF neuron's behavior only up to the point where the potential reaches the threshold. Therefore, it is necessary to simulate the dynamics/compute the threshold crossing to obtain the first spike time of a LIF neuron. The actual firing and reset mechanisms, which affect subsequent dynamics, are not even considered in this simplified formulation. Nevertheless, it turns out that complexity blowup can already occur in this reduced setting.

Theorem 9. *If $P \neq NP$, the LIF model in* (1) *on* $[0, 1]$ *with polynomial-time computable parameters, i.e., coefficients and initial conditions, exhibits complexity blowup.*

Remark 10. *The condition $P \neq NP$ can, in fact, be weakened to $\#P \neq FP$ (noting that equality would also imply $P = NP$). The connection to the complexity classes arises since we show that there exists an input $I_e^* \in Pol([0, 1]; \mathbb{R}^2)$ such that $\mathcal{T}(I_e^*)$ is $\#P$-complete, where $\mathcal{T} : I_e \mapsto (u, I)$ denotes the solution operator of* (1)*. Thus, if the widely accepted conjecture $\#P \neq FP$ holds, it follows that $\mathcal{T}(I_e^*) \neq Pol([0, 1]; \mathbb{R}^2)$. However, the existence proof of I_e^* is non-constructive, i.e., we can neither meaningfully characterize the properties of I_e^* nor identify the subset of inputs with analogous effects on the solution operator.*

This result demonstrates that simulating the dynamics of a LIF neuron on digital hardware can be computationally expensive (or even practically infeasible) when accurate determination of spike times is required. In practice, however, the admissible input signals to networks of LIF neurons are typically more constrained. Specifically, in the formulation given in (2), the input signals are simplified to a decay term and weighted Dirac delta distributions. While this represents an unrealistic theoretical abstraction of the variety of neurophysiological inputs in biological neurons, it nonetheless preserves essential features of their dynamics. Assessing the extent to which this theoretical model retains the advantageous properties of biological systems is therefore an important question, but beyond the scope of this work. Within our complexity framework, we conclude that—in this restricted setting—the input signals themselves may already possess high complexity, which renders the question of output complexity less meaningful, or, conversely, by constraining to a subset of polynomial-time computable input signals, the operator may in fact be polynomial-time computable on this subset, implying that no complexity blowup occurs.

4 Discussion

Related work Understanding the representational power of ANNs has been a central concern in deep learning, which showcased their (universal) capabilities. Similarly, for SNNs, comparable expressivity results have been introduced (Maass 1994, 1997; Zhang and Zhou 2022; Singh, Fono, and Kutyniok 2023; Neuman, Dold, and Petersen 2024). Crucially, they focused on continuous-time models of SNNs, mostly based on the spike response model (Gerstner and van Hemmen 1992), combined with specific coding schemes such as time-to-first-spike coding (Singh, Fono, and Kutyniok 2023; Neuman, Dold, and Petersen 2024) or instantaneous rate coding (Zhang and Zhou 2022), in contrast to our discrete-time perspective, which builds upon the previously discussed work in Nguyen et al. (2025).

Regarding computational complexity of mathematical operations, Friedman (1984) showed that integration exhibits complexity blowup under general conditions. Boche and Pohl (2021) used this result to study complexity blowup in first-order linear ODEs, which we extend in this work to (higher-order) systems of linear ODEs.

Conclusion A deeper understanding of the role of recurrent connections in RLIF-SNNs requires further investigation, e.g., through analyses of input partitioning and the impact of depth. From a theoretical perspective, the advantages of recurrent edges for static data currently appear limited, and their potential power observed in a biological context could not yet be (theoretically) realized in the RLIF-SNN framework. Nonetheless, as with SNNs more broadly, their true potential may emerge in dynamic, event-driven tasks, which still lack rigorous theoretical treatment.

A complementary perspective is provided by the observed complexity blowup in LIF neurons implemented on digital hardware. This result highlights the need for caution when attempting to replicate the computational advantages of biological neural networks on digital platforms. Structural emulation alone may not capture the essential properties of the biological context. Consequently, despite the proven representational power of discrete-time SNNs, their full potential may only be realized through hardware platforms that more closely align with their continuous-time nature.

The difficulty to retain potential features—as exemplarily demonstrated via recurrent connections—and the intricacies arising through simulations—as highlighted by complexity blowup—when converting the continuous-time SNN model into a discrete-time model amenable to digital hardware, motivate the following hypothesis: we need diverse, multi-dimensional models of neurons beyond the pure focus on spikes or/and the implementations of continuous-time SNNs on (analog) neuromorphic hardware may be a crucial step in capturing the advantages of biological networks. This claim, similarly expressed, e.g., in Shen et al. (2024), asks for substantiation in future work.

Acknowledgements

This work of H. Boche was supported in part by the Federal Ministry for Research, Technology and Space of Germany (BMFTR) in the programme of "Souverän.Digital.Vernetzt", joint project 6G-life, project identification number 16KISK002. H. Boche was also partially supported by the project "Next Generation AI Computing (gAIn)", funded by the Bavarian Ministry of Science and the Arts and the Saxon Ministry for Science, Culture, and Tourism.

G. Kutyniok was supported in part by the Munich Center for Machine Learning (BMFTR) as well as the German Research Foundation under Grants DFG-SPP-2298, KU 1446/31-1 and KU 1446/32-1. She also acknowledges support by the Konrad Zuse School of Excellence in Reliable AI (DAAD) and the project "Next Generation AI Computing (gAIn)", which is funded by the Bavarian Ministry of Science and the Arts and the Saxon Ministry for Science, Culture and Tourism.

References

Arora, S.; and Barak, B. 2009. *Computational Complexity: A Modern Approach*. USA: Cambridge University Press, 1st edition. ISBN 0521424267.

Boche, H.; and Pohl, V. 2021. Complexity Blowup in Simulating Analog Linear Time-Invariant Systems on Digital Computers. *IEEE Transactions on Signal Processing*, 69: 5005–5020.

Christensen, D. V.; Dittmann, R.; Linares-Barranco, B.; Sebastian, A.; Le Gallo, M.; Redaelli, A.; Slesazeck, S.; Mikolajick, T.; Spiga, S.; Menzel, S.; Valov, I.; Milano, G.; Ricciardi, C.; Liang, S.-J.; Miao, F.; Lanza, M.; Quill, T. J.; Keene, S. T.; Salleo, A.; Grollier, J.; Markovic, D.; Mizrahi, A.; Yao, P.; Yang, J. J.; Indiveri, G.; Strachan, J. P.; Datta, S.; Vianello, E.; Valentian, A.; Feldmann, J.; Li, X.; Pernice, W. H.; Bhaskaran, H.; Furber, S.; Neftci, E.; Scherr, F.; Maass, W.; Ramaswamy, S.; Tapson, J.; Panda, P.; Kim, Y.; Tanaka, G.; Thorpe, S.; Bartolozzi, C.; Cleland, T. A.; Posch, C.; Liu, S.-C.; Panuccio, G.; Mahmud, M.; Mazumder, A. N.; Hosseini, M.; Mohsenin, T.; Donati, E.; Tolu, S.; Galeazzi, R.; Christensen, M. E.; Holm, S.; Ielmini, D.; and Pryds, N. 2022. 2022 Roadmap on Neuromorphic Computing and Engineering. *Neuromorph. Comput. Eng.*, 2(2). 022501.

Dampfhoffer, M.; Mesquida, T.; Valentian, A.; and Anghel, L. 2024. Backpropagation-Based Learning Techniques for Deep Spiking Neural Networks: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(9): 11906–11921.

Eshraghian, J. K.; Ward, M.; Neftci, E. O.; Wang, X.; Lenz, G.; Dwivedi, G.; Bennamoun, M.; Jeong, D. S.; and Lu, W. D. 2023. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE*, 111(9): 1016–1054.

Friedman, H. 1984. The computational complexity of maximization and integration. *Advances in Mathematics*, 53(1): 80–98.

Gerstner, W.; Kistler, W. M.; Naud, R.; and Paninski, L. 2014. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. USA: Cambridge University Press.

Gerstner, W.; and van Hemmen, J. L. 1992. Associative memory in a network of 'spiking' neurons. *Network: Computation in Neural Systems*, 3(2): 139–164.

Gonzalez, H. A.; Huang, J.; Kelber, F.; Nazeer, K. K.; Langer, T. H.; Liu, C.; Lohrmann, M. A.; Rostami, A.; Schöne, M.; Vogginger, B.; Wunderlich, T.; Yan, Y.; Akl, M.; and Mayr, C. 2023. SpiNNaker2: A Large-Scale Neuromorphic System for Event-Based and Asynchronous Machine Learning. In *Machine Learning with New Compute Paradigms*.

Khalife, S.; Cheng, H.; and Basu, A. 2024. Neural networks with linear threshold activations: structure and algorithms. *Mathematical Programming*, 206(1): 333–356.

Ko, K.-I. 1991. *Complexity theory of real functions*. USA: Birkhauser Boston Inc. ISBN 0817635866.

Maass, W. 1994. On the Computational Complexity of Networks of Spiking Neurons. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 7.

Maass, W. 1997. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9): 1659–1671.

Mehonic, A.; Ielmini, D.; Roy, K.; Mutlu, O.; Kvatinsky, S.; Serrano-Gotarredona, T.; Linares-Barranco, B.; Spiga, S.; Savel'ev, S.; Balanov, A. G.; Chawla, N.; Desoli, G.; Malavena, G.; Monzio Compagnoni, C.; Wang, Z.; Yang, J. J.; Sarwat, S. G.; Sebastian, A.; Mikolajick, T.; Slesazeck, S.; Noheda, B.; Dieny, B.; Hou, T.-H. A.; Varri, A.; Brückerhoff-Plückelmann, F.; Pernice, W.; Zhang, X.; Pazos, S.; Lanza, M.; Wiefels, S.; Dittmann, R.; Ng, W. H.; Buckwell, M.; Cox, H. R. J.; Mannion, D. J.; Kenyon, A. J.; Lu, Y.; Yang, Y.; Querlioz, D.; Hutin, L.; Vianello, E.; Chowdhury, S. S.; Mannocci, P.; Cai, Y.; Sun, Z.; Pedretti, G.; Strachan, J. P.; Strukov, D.; Le Gallo, M.; Ambrogio, S.; Valov, I.; and Waser, R. 2024. Roadmap to neuromorphic computing with emerging technologies. *APL Materials*, 12(10): 109201.

Neuman, A. M.; Dold, D.; and Petersen, P. C. 2024. Stable Learning Using Spiking Neural Networks Equipped With Affine Encoders and Decoders. *arXiv:2404.04549*.

Nguyen, D. A.; Araya, E.; Fono, A.; and Kutyniok, G. 2025. Time to Spike? Understanding the Representational Power of Spiking Neural Networks in Discrete Time. In *Forty-second International Conference on Machine Learning*.

Orchard, G.; Frady, E. P.; Rubin, D. B. D.; Sanborn, S.; Shrestha, S. B.; Sommer, F. T.; and Davies, M. 2021. Efficient Neuromorphic Signal Processing with Loihi 2. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, 254–259.

Pandey, M.; Fernandez, M.; Gentile, F.; Isayev, O.; Tropsha, A.; Stern, A. C.; and Cherkasov, A. 2022. The transformational role of GPU computing and deep learning in drug discovery. *Nat. Mach. Intell.*, 4(3): 211–221.

Pour-El, M. B.; and Richards, J. I. 2017. *Computability in Analysis and Physics*. Perspectives in Logic. Cambridge University Press.

Rathi, N.; Chakraborty, I.; Kosta, A.; Sengupta, A.; Ankit, A.; Panda, P.; and Roy, K. 2023. Exploring Neuromorphic Computing Based on Spiking Neural Networks: Algorithms to Hardware. *ACM Comput. Surv.*, 55(12).

Shen, G.; Zhao, D.; Li, T.; Li, J.; and Zeng, Y. 2024. Are Conventional SNNs Really Efficient? A Perspective from Network Quantization. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 27528–27537.

Silvano, C.; Ielmini, D.; Ferrandi, F.; Fiorin, L.; Curzel, S.; Benini, L.; Conti, F.; Garofalo, A.; Zambelli, C.; Calore, E.; Schifano, S. F.; Palesi, M.; Ascia, G.; Patti, D.; Perri, S.; Petra, N.; Caro, D. D.; Lavagno, L.; Urso, T.; Cardellini, V.; Cardarilli, G. C.; and Birke, R. 2023. A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms. *arXiv:2306.15552*.

Singh, M.; Fono, A.; and Kutyniok, G. 2023. Expressivity of Spiking Neural Networks through the Spike Response Model. In *UniReps: the First Workshop on Unifying Representations in Neural Models*.

Thompson, N. C.; Greenewald, K.; Lee, K.; and Manso, G. F. 2021. Deep Learning's Diminishing Returns: The Cost of Improvement is Becoming Unsustainable. *IEEE Spectrum*, 58(10): 50–55.

Turing, A. M. 1936. On Computable Numbers, with an Application to the Entscheidungs-problem. *Proc. Lond. Math. Soc.*, s2-42(1): 230–265.

Vidal-Saez, M. S.; Vilarroya, O.; and Garcia-Ojalvo, J. 2024. Biological computation through recurrence. *Biochemical and Biophysical Research Communications*, 728: 150301.

Weihrauch, K. 2000. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg. ISBN 9783540668176.

Zhang, S.-Q.; and Zhou, Z.-H. 2022. Theoretically Provable Spiking Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

A Proofs

Proof of Theorem 7

Outline The approximation rates of RLIF-SNNs without explicit recurrent connections for Lipschitz-continuous functions in Nguyen et al. (2025) are derived by a constructive 2-step approach. First, the authors show that continuous functions can be arbitrarily well-approximated by step functions constant on hypercubes. Second, a 2-layer RLIF-SNN with $V^\ell = 0$ is constructed that in the first layer partitions the input space into hypercubes and the second layer assigns values to each hypercube.

By introducing explicit recurrent connections, we try to optimize the construction for continuously differentiable functions by exploiting the fact that RLIF-SNNs can efficiently approximate linear segments. In particular, we employ piecewise linear functions to approximate continuously differentiable functions and then construct an RLIF-SNN approximating the piecewise linear function.

We prove the following more general result that includes Theorem 7 as a special case.

Theorem 11. *Let $f \in \mathcal{C}(U, \mathbb{R}^m)$, $U \subset \mathbb{R}^n$, be such that $f\,|_{U^o} \in \mathcal{C}^1(U^o, \mathbb{R}^m)$ has bounded total derivative df, i.e., $M := \sup_{x \in U^o} \|df\,|_{U^o}\,(x)\|_2 < \infty$. Further, let $\Omega \subset U$ be an arbitrary non-empty subset and $C \subset \mathbb{R}^n$ a half-open cube such that $\Omega \subset C$ and $(\Omega + B_{\rho,2}(0)) \cap C^o \subset U^o$ for a $\rho > 0$.*

For all $\varepsilon > 0$, there exists a RLIF-SNN Φ with $L = 2$ and $T = (K(\mu)+1)T_r(\nu) + 2$ such that

$$\sup_{x \in \Omega} \|R(\Phi)(x) - f(x)\|_2 \leq \varepsilon,$$

where

$$T_r(\nu) := \max\left\{2, \left\lceil \sqrt{n} \frac{\operatorname{diam}_\infty(C)}{K(\mu)} \frac{M}{\nu} \right\rceil \right\},$$

$$K := K(\mu) := \min_{\substack{\xi,\theta>0 \\ \xi\theta=\mu}} \left\{ \left\lceil \sqrt{n} \frac{\operatorname{diam}_\infty(C)}{2\min(\omega^\dagger(\xi), \theta, \frac{\rho}{2})} \right\rceil \right\},$$

with $\nu + \mu = \varepsilon$, $\omega^\dagger$ denoting the generalized inverse of a modulus of uniform continuity of $df\,|_{U^o}$ with respect to $\|\cdot\|_2$, and $\operatorname{diam}_\infty(C) := \sup_{x,y \in C} \|x - y\|_\infty$. Moreover, the width parameter $n = (n_1, n_2)$ are given by

$$n_1 = n + 1,$$
$$n_2 = K(\mu)^n (n+1) + 3.$$

Remark 12. *Let $f : C \to \mathbb{R}^m$ be defined on a half-open cube $C \subset \mathbb{R}^n$ and $\Omega = C$. We can then choose ρ arbitrarily large, i.e., we can drop $\frac{\rho}{2}$ from the definition of K, to obtain*

$$K(\mu) = \min_{\substack{\xi,\theta>0 \\ \xi\theta=\mu}} \left\{ \left\lceil \sqrt{n} \frac{\operatorname{diam}_\infty(C)}{2\min(\omega^\dagger(\xi), \theta)} \right\rceil \right\}.$$

Thus, we recovered the statement in Theorem 7.

Remark 13. *The obtained approximation rates are not generally improved in comparison with the ones presented in Nguyen et al. (2025) for RLIF-SNNs without explicit recurrent connections. A concrete counter example is a sinus wave with high frequency and small amplitude: $f(x) = \frac{\sin(nx)}{n}$ with $n \in \mathbb{N}$ on $C = U = [0, 3\phi)$ and $\Omega = [0, 2\phi]$. Hence, although for certain functions the incorporation of recurrent connections may indeed be beneficial, we could not show a general improvement with our construction.*

Definition and Notation We define the modulus of uniform continuity $\omega : [0, \infty] \to [0, \infty]$ for a uniformly continuous function $f : M \to N$ on metric spaces M, N such that $\lim_{x \to 0} \omega(x) = 0$ and

$$d_N(f(x), f(y)) \leq \omega(d_M(x, y)) \quad \forall x, y \in M.$$

Its generalized inverse of a modulus of uniform continuity $\omega^\dagger$ is then given by

$$\omega^\dagger(s) = \inf\{t \in [0, \infty] \mid \omega(t) > s.$$

We also write $\|f\|_{\infty,p} := \sup_{x \in U} \|f(x)\|_p$ for a function $f : U \to \mathbb{R}^m$.

Auxiliary Results We first observe that continuous differentiable functions with a uniform continuous derivative can be efficiently approximated by piecewise linear functions.

Lemma 14. *Let K be defined as in Theorem 11. Let $f \in \mathcal{C}(U, \mathbb{R}^m)$, $U \subset \mathbb{R}^n$, be a continuous function such that $f\,|_{U^o} \in \mathcal{C}^1(U^o, \mathbb{R}^m)$ is continuously differentiable with uniformly continuous total derivative $df|_{U^o}$. Further, let $\Omega \subset U$ be an arbitrary non-empty subset and $C \subset \mathbb{R}^n$ a half-open cube such that $\Omega \subset C$ and $(\Omega + B_{\rho,2}(0)) \cap C^o \subset U^o$ for a $\rho > 0$.*

For every $\mu > 0$, we can decompose C into K^n half-open subcubes $(C^{(j)})_{j\in[K^n]}$ such that affine linear functions $g^{(j)} : C^{(j)} \to \mathbb{R}^m$ exist with $\|d(g^{(j)})\|_{\infty,2} \leq \|df|_{U^o}\|_{\infty,2}$ and $\|(f-g)|_\Omega\|_{\infty,2} < \mu$, where $g = \sum_{i=1}^m g^{(j)}\chi_{C^{(j)}}$.

Proof. The result follows from considering $g^{(j)} : C^{(j)} \to \mathbb{R}^m$ defined as

$$g^{(j)} = \begin{cases} x \mapsto f(c^{(j)}) + df_{c^{(j)}}(x - c^{(j)}), & C^{(j)} \cap \Omega \neq \emptyset \\ x \mapsto 0, & C^{(j)} \cap \Omega = \emptyset \end{cases},$$

where $c^{(j)}$ is the center of $C^{(j)}$. □

Next, we provide an alternative characterization of RLIF-SNN dynamics that avoids (implicit) recurrence.

Definition 15. *Let $t \in [T]$, $l \in [L]$ and spike train families $\sigma = (\sigma^{[l']})_{l'\in\{0,\dots,l\}}$, $\sigma' = (\sigma'^{[l']})_{l'\in\{0,\dots,l\}}$ be given, such that $\forall_{l\in\{0,\dots,l-1\}}\sigma^{[l']} \in \{0,1\}^{n_{l'}\times t}$ and $\sigma^{[l]} \in \{0,1\}^{n_l\times t}$ as well as $\forall_{l'\in\in\{0,\dots,l\}}\sigma'^{[l']} \in \{0,1\}^{n_{l'}\times t}$, i.e. σ, σ' have to be chosen such that the terms in the following definitions are well-defined. We define*

$$\begin{aligned}
I^{[l]}(t;\sigma) &:= (\alpha^{[l]})^t i^{[l]}(0) \\
&\quad + \sum_{k=1}^{t} (\alpha^{[l]})^{t-k} \left(W^{[l]}\sigma^{[l-1]}(k) + V^{[l]}\sigma^{[l]}(k-1) \right), \\
p^{[l]}(t;\sigma) &:= (\beta^{[l]})^t u^{[l]}(0) + \sum_{k=1}^{t} (\beta^{[l]})^{t-k} \left(I^{[l]}(k;\sigma) + b^{[l]} \right) \\
&\quad - \vartheta \sum_{k=1}^{t-1} (\beta^{[l]})^{t-k}\sigma^{[l]}(k), \\
S^{[l]}(t;\sigma) &:= H(p^{[l]}(t;\sigma) - \vartheta \mathbf{1}_{n_l}), \\
u^{[l]}(t;\sigma') &:= (\beta^{[l]})^t u^{[l]}(0) \\
&\quad + \sum_{k=1}^{t} (\beta^{[l]})^{t-k} \left(I^{[l]}(k;\sigma') + b^{[l]} - \vartheta\sigma'^{[l]}(k) \right).
\end{aligned}$$

One verifies by direct computation that the non-recursive formulas are equivalent to the recursive formulation introduced in Definition 1.

Lemma 16. *The non-recursive formulas from Definition 15 are equivalent to the recursive definitions for $l \in [L]$, $t \in [T]$ assuming previous spikes are equal, i.e., $\forall_{l'\in\{0,\dots,l-1\}}\sigma^{[l']} = S^{[l']}$, $\forall_{t'\in[t-1]}\sigma^{[l]}(t') = S^{[l]}(t')$, and $\forall_{l'\in\{0,\dots,l\}}\sigma'^{[l']} = S^{[l']}$.*

Proof of Theorem 11 Let $\varepsilon, \mu, \nu > 0$ with $\varepsilon = \mu + \nu$. By Lemma 14, we can obtain a decomposition of C into K^n half-open subcubes $(C^{(j)})_{i=1..K^n}$ and linear functions $g^{(j)} : C^{(j)} \to \mathbb{R}^m$, such that $\|d(g^{(j)})\|_{\infty,2} \leq \|df\|_{\infty,2}$ and $\|f - g\|_{\infty,2} < \mu$ for $g = \sum_{i=1}^m g^{(j)}\chi_{C^{(j)}}$. Next, we will define an RLIF-SNN Φ such that $\|R(\Phi)|_C - g\|_{\infty,2} < eta$. First, we fix $I^{[l]}(0) = 0$, $\alpha = 0$, and $\beta = 1$ for all layers.

In our construction, we use the following five phases:

$$T_1 = \{1, \dots, KT_r\}, T_2 = \{KT_r\}, T_3 = \{KT_r + 1\},$$
$$T_4 = \{KT_r + 2\}, T_5 = \{KT_r + 3, \dots, T\}.$$

Note that we will occasionally refer to T_i as numbers for ease of notation. The high-level idea is that we accumulate the potential during T_1, determine the subregion $C^{(j)}$ of the input domain during $T_2, \dots, T_4$, and compute the position of the input inside of $C^{(j)}$ during T_5. The first layer will only be active during the first phase T_1. It is composed of $n + 1$ neurons, where the first n neurons convert the input vector based on its position in C into spike trains. The last neuron, the 'alarm neuron', shuts down the first layer after T_1 ends. The second layer only accumulates potential without spikes during $T_1 \setminus T_2$. Then during $T_2 \cup T_3 \cup T_4$, the input x is located in the region $C^{(j)}$ and subsequently during $T_4 \cup T_5$ the location inside $C^{(j)}$ is encoded through spikes. For each region $C^{(j)}$, we have $n + 1$ neurons in the second layer. Hereby, each of the first n neurons encodes a component of the linear part of $g^{(j)}$. They are also used to inform the $n + 1$-th neuron of the group if the input x is at least as big as the base point of $C^{(j)}$. Finally, the $n + 1$-th neuron deactivates all other neurons of regions with a smaller base point and encodes the constant part of $g^{(j)}$. The last 3 neurons act as 'clock neurons', enabling and disabling the other neurons in the layer. Hence, explicit recurrent connections are necessary for our clock and alarm neurons as well as 'control' neurons.

Now we present the construction in more detail, which leads to the described behaviour. Thereby, to obtain the normalized location of a value in $C = \prod_{i=1}^n [x_i, y_i)_{i=1}$ we will often

$$o_i(z) = \frac{z_i - x_i^C}{y_i^C - x_i^C}, i \in \mathbb{N}.$$

First layer: We define the i-th neuron, $i = 1, \dots, n$, in the first layer by

$$w = \frac{1}{y_i^C - x_i^C} e_i, b = -\frac{x_i^C}{y_i^C - x_i^C}, v = -e_{a_1}, u_0 = 0. \quad (i)$$

The 'alarm neuron' of the first layer, with index $a_1 = +1$, is defined by:

$$w = 0, b = \frac{1}{T_2}, v = e_{a_1}, u_0 = 0. \quad (a_1)$$

Second layer: For each of the K^n subcubes in C we define $n + 1$ neurons in the following way: Let $C^{(j)} = \prod[x^{C^{(j)}}, y^{C^{(j)}})$ be a subcube with position $q^{(j)} \in \{0, \dots, K^n - 1\}$ in C, i.e.

$$q_i^{(j)} = Ko_i(x^{C^{(j)}}) \quad \forall i \in [n].$$

We will write $\iota_j(i) = j(n+1)+i$ to index the first n neurons in the layer and $\omega_j = (j+1)(n+1)$ to index the last neuron of each group.

The i-th neuron of the first n neurons (of the j-th group), with index $\iota_j(i)$ in the second layer, has the parameters

$$w = e_i, b = 0, v = T(e_{c_1} - 2e_{c_2} + r(q^{(j)})),$$
$$u_0 = -q_i^{(j)} T_r - T + 1, \qquad (\iota_j(i))$$

where the 'switch' is

$$r(q) = e_{\omega_{j(q)}} - \sum_{\substack{q' \in \{0,\dots,K^n-1\} \\ q<q'}} e_{\omega_{j(q')}}$$

with the index $j(q)$ of the subcube at position q. We further define the applied variant

$$r_S(q;t) = \langle r(q), S^{[2]}(t) \rangle = S^{[2]}_{\omega_{j(q)}}(t) - \sum_{\substack{q' \in \{0,\dots,K^n-1\} \\ q<q'}} S^{[2]}_{\omega_{j(q')}}(t).$$

The final neuron of the group, with index ω_j in its layer, has the parameters

$$w = 0, b = 0, v = \frac{1}{n}\sum_{i=1}^{n} e_{\iota_j(i)} - 2e_{a_2} + r(q^{(j)}),$$
$$u_0 = 0. \qquad (\omega_j)$$

We also define the two 'clock neurons', with index $c_1 = (n+1)K^n + 1$ and $c_2 = (n+1)K^n + 2$ with parameters:

$$w = 0, b = b_{c_i}, v = -(T-1)e_{c_i}, u_0 = 0, \qquad (c_1, c_2)$$

where $b_{c_1} = \frac{1}{T_2 - 1}$ and $b_{c_2} = \frac{1}{T_2}$. We further define the 'alarm neuron', with index $a_2 = (j+1)K^n + 3$, by

$$w = 0, b = \frac{1}{T_4}, v = e_{a_2}, u_0 = 0. \qquad (a_2)$$

Output decoder: We set the parameters of the output decoder to $a_t = 0$, for $t \leq T_3$ and $a_t = 1$ otherwise. We further fix $b^{[L+1]} = 0$ and

$$W^{[L+1]}_{k,\iota_j(i)} = d(g^{(j)})_k((y_i^{C^{(j)}} - x_i^{C^{(j)}})\frac{1}{T_r}e_i)$$

for $k \in [m]$, $j \in [K^n]$, and $i \in [n]$. Note that $d(g^{(j)})$, as total derivative, represents the linear part of $g^{(j)}$, i.e. $g^{(j)}(x) = d(g^{(j)})(x - x^{C^{(j)}}) + g(x^{C^{(j)}})$ for all $x \in C^{(j)}$. We further set

$$W^{[L+1]}_{k,\omega_j} = g_k(x^{C^{(j)}})$$

for $k \in [m]$ and $j \in [K^n]$. We finally define $W^{[L+1]}_{k,c_i} = 0$ for $i \in \{1, 2\}$.

We will now show that this construction approximates g well enough, i.e., for any $x = S^{[0]}(t) \in C$ we have $\|R(\Phi)(x) - g(x)\|_{\infty,2} \leq \nu$. Note that with the chosen parameters, the dynamics simplify to

$$I^{[l]}(t) = W^{[l]}S^{[l-1]}(t) + V^{[l]}S^{[l]}(t-1)$$
$$p^{[l]}(t) = u^{[l]}(t-1) + I^{[l]}(t) + b^{[l]}$$
$$S^{[l]}(t) = H(p^{[l]}(t) - \mathbf{1}_{n_l})$$
$$u^{[l]}(t) = p^{[l]}(t) - S^{[l]}(t)$$

and via the non-recursive formula in Lemma 16 we get

$$p^{[l]}(t) = u^{[l]}(0) + \sum_{k=1}^{t}\left(I^{[l]}(t) + b^{[l]}\right) - \sum_{k=1}^{t-1} S^{[l]}(k).$$

Characterizing the **first layer**:

- 'alarm neuron' a_1: One verifies via $p^{[1]}_{a_1}(t)$ that $S^{[1]}_{a_1}(t) = 1 \iff t \geq T_2$

$$p^{[1]}_{a_1}(t) = \frac{t}{T_2} + \sum_{k=1}^{t} s^{[1]}_{a_1}(k-1) - \sum_{k=1}^{t-1} s^{[1]}_{a_1}(k) = \frac{t}{T_2}$$

- i-th neuron, $i \in [n]$: We obtain

$$\lfloor T_2 o_i(x) \rfloor = \left\lfloor u_i^{[1]}(0) + \sum_{t=1}^{T_2}(I_i^{[1]}(t) + b_i^{[1]}) \right\rfloor = \sum_{t=1}^{T_2} S_i^{[1]}(t)$$

and $S_i^{[1]}(t) = 0$ for $t > T_2$.

Characterizing the **second layer**:

- 'clock neurons': We have $S^{[2]}_{c_1}(t) = 0$ for all $t \in T \setminus \{T_2 - 1\}$ and $S^{[2]}_{c_1}(T_2 - 1) = 1$. Similarly we get $S^{[2]}_{c_1}(t) = 0$ for all $t \in T \setminus \{T_2\}$ and $S^{[2]}_{c_2}(T_2) = 1$
- 'alarm neuron': As for the alarm neuron in the first layer, we obtain $S^{[1]}_{a_2}(t) = 1 \iff t \geq T_4$.

The behavior of the remaining neurons in the second layer will be tracked throughout the phases.

- **Phase 1** (T_1): One verifies that $S^{[2]}_{\iota_j(i)}(t) = 0$ and $S^{[2]}_{\omega_j}(t) = 0$ for all for all $i \in [n]$, $j \in [K^n]$ and $t \in \{0, \dots, T_2 - 1\}$ by induction over t.
- **Phase 2** (T_2): As in the first phase, we get $S^{[2]}_{\omega_j}(T_2) = 0$. Similarly, one checks that $S^{[2]}_{\iota_j(i)}(T_2) = 1$ if and only if $x_i^{C^{(j)}} \leq x_i$.
- **Phase 3** (T_3): We have $S^{[2]}_{\omega_j}(T_3) = 1$ if and only if $x_i^{C^{(j)}} \leq x_i$ for all $i \in [n]$. Moreover, $S^{[2]}_{\iota_j(i)}(T_3) = 0$ for all $i \in [n]$ holds as well.
- **Phase 4** (T_4): Similarly, $S^{[2]}_{\omega_j}(T_4) = 1$ if and only if $x \in C^{(j)}$ and $S^{[2]}_{\iota_j(i)}(T_4) = 0$ for all $i \in [n]$.
- **Phase 5** (T_5): On the one hand, $S^{[2]}_{\omega_j}(t) = 0$ for all $t > T_4$, and, conversely, the neuron $\iota_j(i)$ captures the position of x in $C^{(j)}$ regarding the i-th dimension if $x \in C^{(j)}$ and stays inactive otherwise, i.e., for $x \notin C^{(j)}$ we have $S^{[2]}_{\iota_j(i)}(t) = 0$ for all $t > T_4$ whereas for j such that $x \in C^{(j)}$ we obtain

$$\sum_{t=T_4+1}^{T} S^{[2]}_{\iota_{j(i)}}(t) = \left\lfloor u^{[2]}_{\iota_{j(i)}}(T_4) + I^{[2]}_{\iota_{j(i)}}(T_4+1) + b^{[2]}_{\iota_{j(i)}} \right\rfloor$$
$$= -KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor.$$

Overall, we have shown that

- the group of neurons $\iota_{j(i)}$ and ω_j in the second layer does not fire, i.e., does not contribute to the output of the network, provided that $x \notin C^{(j)}$,
- for j such that $x \in C^{(j)}$ we get

$$W^{[L+1]}_{k,\omega_j} \sum_{t=T_4}^{T} S^{[2]}_{\omega_j}(t) = g_k(x^{C^{(j)}}),$$

$$W^{[L+1]}_{k,\iota_{j(i)}} \sum_{t=T_4}^{T} S^{[2]}_{\iota_{j(i)}}(t) = \left(-KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor\right) \cdot (g^{(j)})_k(y_i^{C^{(j)}} - x_i^{C^{(j)}}) \frac{1}{T_r} e_i.$$

for all $k \in [m]$ by choice of the decoder parameter.

Moreover, again by choice of the decoder parameter, the neurons c_1, c_2, a_2 do not contribute to the output as well, so that

$$\begin{aligned} R(\Phi)_k(x) &= \sum_{t=T_4}^{T} (W^{[L+1]} S^{[2]}(t))_k \\ &= W^{[L+1]}_{k,\omega_j} \sum_{t=T_4}^{T} S^{[2]}_{\omega_j}(t) + W^{[L+1]}_{k,\iota_{j(i)}} \sum_{t=T_4}^{T} S^{[2]}_{\iota_{j(i)}}(t) \\ &= g_k(x^{C^{(j)}}) + \sum_{i\in[n]} \left(d(g^{(j)})_k((y_i^{C^{(j)}} - x_i^{C^{(j)}}) \frac{1}{T_r} e_i) \right) \cdot \\ &\quad \left(-KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor\right) \\ &= g_k(x') \end{aligned}$$

for

$$x' = x^{C^{(j)}} + \sum_{i\in[n]} \frac{y_i^{C^{(j)}} - x_i^{C^{(j)}}}{T_r} (\lfloor KT_r o_i(x) \rfloor - KT_r o_i(x^{C^{(j)}})) e_i.$$

Now, by assumption, we have

$$T_r = \frac{\sqrt{n}(y_i^C - x_i^C)}{K2\nu} \|df\|_{\infty,2} \geq \frac{\sqrt{n}(y_i^{C^{(j)}} - x_i^{C^{(j)}})}{2\nu} \|d(g^{(j)})\|_{\infty,2}$$

for any $i \in [n]$ and hence

$$\xi_i := \frac{1}{T_r} \left(\lfloor KT_r o_i(x) \rfloor - KT_r o_i(x) \right) \leq \frac{2\nu \|d(g^{(j)})\|^{-1}_{\infty,2}}{\sqrt{n}(y_i^{C^{(j)}} - x_i^{C^{(j)}})}.$$

Thus, we compute

$$\begin{aligned} \|x' - x\|_2^2 &= \sum_{i\in[n]} (x' - x_i)^2 = \sum_{i\in[n]} \xi_i^2 (y_i^{C^{(j)}} - x_i^{C^{(j)}})^2 \\ &\leq \frac{\nu^2}{\|d(g^{(j)})\|^2_{\infty,2}} \end{aligned}$$

and conclude

$$\begin{aligned} \|g(x) - R(\Phi)_k(x)\|_2 &= \|g(x) - g(x')\|_2 \\ &= \|d(g^{(j)})(x - x')\|_2 \\ &\leq \|d(g^{(j)})\|_{\infty,2} \|x - x'\|_2 \leq \nu. \end{aligned}$$

Proof of Theorem 9

Outline We first introduce necessary notation, followed by some auxiliary results. Subsequently, we present our main theorem and its proof about complexity blowup in systems of linear ODEs, and finally apply it to derive our findings about complexity blowup in LIF neurons.

Definition and Notation To complement the definitions of computable objects in the main section, we employ the following notation. We write $\mathbb{R}_c$, $\mathbb{C}_c$, and $\mathbb{C}_c^{m\times n}$ for the set of computable real numbers, computable complex numbers, and computable complex matrices, respectively. Analogously, we write $\mathbb{R}_p$, $\mathbb{C}_p$, and $\mathbb{C}_p^{m\times n}$ for the set of polynomial-time computable real numbers, complex numbers, and complex matrices, respectively. In both cases, (polynomial-time) computability is to be understood component-wise so that it reduces to the introduced definition of (polynomial-time) computability of real numbers.

Next, we formalize for clarity the type of ODEs we consider in the remainder.

Definition 17 (Linear Ordinary Differential Equation)**.** *A* linear ordinary differential equation *of order* $(m, n) \in \mathbb{N}^2$ *is of the form*

$$\sum_{i=0}^{m} a_i y^{(i)} = \sum_{k=0}^{n} b_k x^{(k)},$$

where $a_i, b_j \in \mathbb{C}$, $a_m, b_n \neq 0$ *are constants,* $x \in \mathcal{C}^{n+1}([0,1])$ *is the inhomogeneous generator and* $y \in \mathcal{C}^m([0,1])$ *is the unknown function. Since* $a_m \neq 0$*, we may always assume without loss of generality that* $a_m = 1$*, and we write* $\mathcal{T}_{m,n}((a_i)_{i=0}^m, (b_k)_{k=0}^n) : x \mapsto y$ *for the solution operator.*

We will express our main result about the complexity blowup of linear ODEs in terms of the properties of the characteristic polynomials, which we introduce next.

Definition 18 (Characteristic Polynomial)**.** *Consider a linear ODE of order* (m, n)*, i.e. of the form*

$$\sum_{i=0}^{m} a_i y^{(i)} = \sum_{k=0}^{n} b_k x^{(k)}.$$

Then we define the characteristic polynomials $P_y(X)$ *and* $P_x(X)$ *of the ODE as*

$$P_y(X) = \sum_{i=0}^{m} a_i X^i \in \mathbb{C}[X] \text{ and}$$

$$P_x(X) = \sum_{k=0}^{n} b_k X^k \in \mathbb{C}[X].$$

To analyze the properties of polynomials, in particular the previously described characteristic polynomials, we specify the following notation. We denote by $Q \mid P$ the divisibility of a polynomial $P \in R[X]$, where R is a ring, by another polynomial $Q \in R[X]$, i.e., $P = Q \cdot R$ for some polynomial $R \in R[X]$. Moreover, we write $\mathcal{Z}(P)$ for the multi-set of all roots of P.

We briefly extend our framework to systems of linear ODEs that essentially couple several scalar ODEs.

Definition 19 (System of linear ODE and its characteristic polynomial)**.** *A d-system of linear ODEs of order $(m,n) \in \mathbb{N}^2$ is of the form*

$$\sum_{i=0}^{m} A_i \cdot y^{(i)} = \sum_{j=0}^{n} B_j \cdot x^{(j)}$$

where $A_i, B_j \in \mathbb{C}^{d\times d}$ are constant, $x \in \mathcal{C}^{n+1}([0,1])^d$ is the inhomogeneous generator and $y \in \mathcal{C}^m([0,1])^d$ is the unknown function. We write $\mathcal{T}^d_{m,n}((A_i)_{i=0}^m, (B_j)_{j=0}^n) : x \mapsto y$ for the solution operator and $P_y(X)$, $P_x(X)$ for its characteristic polynomials *given by*

$$P_y(X) = \sum_{i=0}^{m} A_i X^i \in \mathbb{C}^{d\times d}[X] \text{ and}$$
$$P_x(X) = \sum_{j=0}^{n} B_j X^j \in \mathbb{C}^{d\times d}[X].$$

Finally, we define the notion of a polynomial-time computable ODE based on its parameters, which is the setting we consider to analyze complexity blowup.

Definition 20. *A (system of) linear ODE(s) of order (m,n) is called* polynomial-time computable *if all coefficients and the initial conditions are polynomial-time computable.*

Main Result Now, we are ready to state our main result about complexity blowup in linear ODEs.

Theorem 21. *If $FP \neq \#P$, a polynomial-time computable linear ODE of order (m,n) exhibits complexity blowup if and only if $P_y \nmid P_x$.*

Before proving the theorem in the remainder of this section, we describe the extension of the result to systems of ODEs under some conditions.

Corollary 22. *If $FP \neq \#P$, a polynomial-time computable d-system of linear ODE of order $(1,n)$*

$$A_1 y' + A_0 y = \sum_{j=0}^{n} B_j x^{(j)}$$

with A_1 invertible and for $j = 0,\dots,n$

$$[A_1^{-1}A_0, A_1^{-1}B_j] := A_1^{-1}A_0A_1^{-1}B_j - A_1^{-1}B_jA_1^{-1}A_0 = 0$$

exhibits complexity blowup if and only if $P_x(-A_1^{-1}A_0) \neq 0$.

Proof. Under the given assumptions, the proof is completely analogous to the single case treated in Theorem 21, in particular, the step in Proposition 27. Indeed, the given conditions account for the facts that $\mathbb{C}^{d\times d}[X]$, respectively $\mathbb{C}_p^{d\times d}[X]$, are not algebraically closed and commutative, but we still are able to mirror the single ODE proof strategy. □

Auxiliary Results The set $\mathbb{C}_p$ is an algebraically closed field and $\mathbb{R}_p \subset \mathbb{C}_p$ is a subfield (Pour-El and Richards 2017). Based on this structure, we collect some useful properties about the complexity of functions and operations employed in the subsequent analysis (Pour-El and Richards 2017; Ko 1991).

Proposition 23.

- *Let $\mathbb{A} \in \{\mathbb{R}, \mathbb{C}\}$. Then we have*
$$\{x + y \mid x \in \mathbb{A}_p, y \in \mathbb{A}_p^C\} \subseteq \mathbb{A}_p^C \text{ and}$$
$$\{x \cdot y \mid x \in \mathbb{A}_p^*, y \in \mathbb{A}_p^C\} \subseteq \mathbb{A}_p^C,$$
where $\mathbb{A}_p^*$ denotes the set of elements with a (multiplicative) inverse in $\mathbb{A}_p$.
- *The set Pol$([a,b])$ is a $\mathbb{C}_p$-algebra.*
- *Let $f \in$ Pol$([a,b]) \cap \mathcal{C}^{n+1}([a,b])$. Then for all $i \leq [n]$ we have $f^{(i)} \in$ Pol$([a,b])$.*
- *Let $z \in \mathbb{C}_p$. Then $e^{z\cdot} \in$ Pol$([0,1])$.*
- *Let $f : [0,1] \to \mathbb{C}$ be $\#P$-complete, let $g \in$ Pol$([0,1])$ and $0 \neq z \in \mathbb{C}_p$ be polynomial-time computable. Then the function*
$$h(t) := zf(t) + g(t)$$
is also $\#P$-complete.

To characterize linear ODEs, we already introduced their characteristic polynomials in Definition 18. To motivate the appearance of these characteristic polynomials more formally, consider a linear differential operator of order n with constant coefficients

$$D : \{f : I \to \mathbb{C} \mid f \text{ is } n\text{-times diff'able}\} \to \{f : I \to \mathbb{C}\},$$
$$f \mapsto \sum_{k=0}^{n} a_k f^{(k)}(x), \quad I \subset \mathbb{R}, a_k \in \mathbb{C}, n \in \mathbb{N}.$$

We write $\mathcal{P}(I)$ and $\mathcal{P}_n(I)$ for the set of all differential operators with constant coefficients on I and those of order n, respectively. Now we can associate to each $D \in \mathcal{P}_n(I)$ a specific polynomial $P_D \in \mathbb{C}[X]$:

$$D = \sum_{k=0}^{n} a_k \frac{d^k}{dx^k} \implies P_D(X) = \sum_{k=0}^{n} a_k X^k.$$

By abuse of notation, we will often write $D = P_D(\frac{d}{dx})$ for simplicity so that a linear ODE as introduced in Definition 17 can be expressed via its characteristic polynomials as

$$P_y\left(\frac{d}{dx}\right) y = P_x\left(\frac{d}{dx}\right) x.$$

Moreover, one easily verifies the following equalities, which we will repeatedly apply in the remainder: For $P, Q \in \mathbb{C}[X]$ we have

$$(P+Q)\left(\frac{d}{dx}\right) = P\left(\frac{d}{dx}\right) + Q\left(\frac{d}{dx}\right) \text{ and}$$
$$(P\cdot Q)\left(\frac{d}{dx}\right) = P\left(\frac{d}{dx}\right) \cdot Q\left(\frac{d}{dx}\right).$$

Another tool we need for our analysis is the reduction of the degree of polynomials. For $y \in R$ and a polynomial $P \in R[X]$ over a ring R with

$$P(X) = \sum_{i=0}^{m} a_i X^i, \quad a_i \in R,$$

we set

$$\mathcal{R}_y[P](X) := \sum_{i=0}^{m-1} b_i(y)X^i \in R[X], \quad \text{where}$$

$$b_i(y) := \begin{cases} a_m, & \text{if } i = m-1 \\ a_{i+1} + y \cdot b_{i+1}(y), & \text{otherwise} \end{cases}. \quad (5)$$

In particular, $\mathcal{R}_y[P](X)$ is the polynomial obtained through the polynomial division of $P(X)$ by $(X-y)$ with remainder term $b_{-1}(y) := a_0 + y \cdot b_0(y)$, i.e.,

$$P(X) = (X-y) \cdot \mathcal{R}_y[P](X) + b_{-1}(y).$$

In this setting, we immediately obtain the following characterizations.

Proposition 24. *Let F be a field and let $P \in F[X]$ be a polynomial.*

- *For $\sigma \in F$, the following are equivalent (with the notation introduced in the previous paragraph): (i) $\sigma \in \mathcal{Z}(P)$, (ii) $b_{-1}(\sigma) = 0$, (iii) $\sigma b_0(\sigma) = -a_0$.*
- *For $\sigma \in \mathcal{Z}(P)$ we have $\mathcal{Z}(\mathcal{R}_\sigma[P]) = \mathcal{Z}(P) \setminus \{\sigma\}$.*

Proof of Theorem 21 The proof of Theorem 21 is essentially separated into two steps. The case of linear ODEs of order (1,1) was previously treated in Boche and Pohl (2021). However, the proof strategy also extends to linear ODEs of order (1,n) with some necessary modifications; the main idea is to rely on the explicit solution formula together with a well-known result about a $\#P$-complete problem.

Proposition 25. *The solution of linear ODE of order $(1,n)$, i.e. of the form*

$$y' + a_0 y = \sum_{k=0}^{n} b_k x^{(k)},$$

is given by

$$y(t) = e^{-a_0 t}y(0) + \sum_{j=0}^{n} \int_0^t b_j e^{-a_0(t-\tau)} x^{(j)}(\tau) d\tau.$$

Theorem 26 (Friedman (1984))**.** *There exists a smooth polynomial-time computable function $f : [0,1] \to \mathbb{C}$ such that the function g given by*

$$g(t) = \int_0^t f(\tau) d\tau$$

is $\#P$-complete.

This allows us to encode a $\#P$-complete problem into the solution of linear ODEs of order $(1,n)$ and thereby obtain the following characterization of its complexity.

Proposition 27. *For any (1,n)-order polynomial-time computable ODE of the form*

$$y' + a_0 y = \sum_{j=0}^{n} b_j x^{(j)}$$

there exists $x \in \mathcal{C}^\infty([0,1]) \cap \mathit{Pol}([0,1])$ such that $\mathcal{T}_{1,n}(a_0, (b_j)_{j=0}^n)(x)$ is $\#P$-complete if and only if $P_x(-a_0) \neq 0$.

Proof. According to Theorem 26, there exists a function $\tilde{f} \in \text{Pol}([0,1]) \cap \mathcal{C}^\infty([0,1])$ such that

$$\tilde{g}(t) := \int_0^t \tilde{f}(\tau) d\tau$$

is $\#P$-complete. Set $x(t) := e^{-a_0 t}\tilde{f}(t)$ and observe that $x \in \text{Pol}([0,1]) \cap \mathcal{C}^\infty([0,1])$ as well via Proposition 23. Since for $i \in \{1, \dots, n\}$ we have

$$\begin{aligned} x^{(i)}(t) &= \sum_{j=0}^{i} \binom{i}{j} (-a_0)^{i-j} e^{-a_0 t} \tilde{f}^{(j)}(t) \\ &= e^{-a_0 t} \sum_{j=0}^{n} \binom{i}{j} (-a_0)^{i-j} \tilde{f}^{(j)}(t), \end{aligned}$$

it follows that

$$\begin{aligned} &\sum_{i=0}^{n} \int_0^t b_i e^{-a_0(t-\tau)} x^{(i)}(\tau)\, d\tau \\ &= \sum_{i=0}^{n} \int_0^t b_i e^{-a_0 t} \sum_{j=0}^{n} \binom{i}{j} (-a_0)^{i-j} \tilde{f}^{(j)}(\tau)\, d\tau \\ &= e^{-a_0 t} \sum_{j=0}^{n} \left(\sum_{i=0}^{n} \binom{i}{j} (-a_0)^{i-j} b_i \right) \int_0^t \tilde{f}^{(j)}(\tau)\, d\tau \\ &= e^{-a_0 t} \sum_{j=0}^{n} \lambda(j) \int_0^t \tilde{f}^{(j)}(\tau)\, d\tau \\ &= e^{-a_0 t} \left(\sum_{j=1}^{n} \lambda(j) f^{(j-1)}(t) + \lambda(0)\tilde{g}(t) \right) \\ &= e^{-a_0 t} \sum_{j=0}^{n-1} \lambda(j+1) f^{(j)}(t) + e^{-a_0 t}\lambda(0)\tilde{g}(t) \end{aligned}$$

with $\lambda : \mathbb{N} \to \mathbb{C}_p$ given by

$$\lambda(j) := \sum_{i=0}^{n} \binom{i}{j} (-a_0)^{i-j} b_i.$$

Thus, by the solution formula in Proposition 25 we obtain that

$$\begin{aligned} \mathcal{T}_{1,n}(a_0, (b_j)_{j=0}^n)(x) &= e^{-a_0 t}y(0) + \sum_{i=0}^{n} \int_0^t b_i e^{-a_0(t-\tau)} x^{(i)}(\tau)\, d\tau \\ &= e^{-a_0 t}y(0) + e^{-a_0 t} \sum_{j=0}^{n-1} \lambda(j+1) f^{(j)}(t) + e^{-a_0 t}\lambda(0)\tilde{g}(t). \end{aligned}$$

Noting that Proposition 23 implies that

$$e^{-a_0 t}\lambda(0),\ e^{-a_0 t} \sum_{j=0}^{n-1} \lambda(j+1) f^{(j)}(t),\ e^{-a_0 t}y(0) \in \text{Pol}([0,1])$$

gives again via Proposition 23 that $\mathcal{T}_{1,n}(a_0, (b_j)_{j=0}^n)(x)$ is $\#P$-complete if

$$0 \neq \lambda(0) = \sum_{i=0}^{n} \binom{i}{0} (-a_0)^{i-0} b_i = \sum_{i=0}^{n} b_i (-a_0)^i = P_x(-a_0).$$

Hence, it remains to show that if $P_x(-a_0) = 0$, then the solution is always polynomial-time computable. Observe that by definition, P_y is a polynomial of degree one with a single root $-a_0$. Therefore, assuming $P_x(-a_0) = 0$ implies that $\{-a_0\} = \mathcal{Z}(P_y) \subset \mathcal{Z}(P_x)$, which by Lemma 28 is equivalent to $P_y \mid P_x$. Thus, there exists $Q \in \mathbb{C}[X]$ with $P_x(X) = P_y(X)Q(X)$ so that the ODE can be rewritten as

$$P_y\left(\frac{d}{dt}\right) y = P_x\left(\frac{d}{dt}\right) x = (P_y \cdot Q)\left(\frac{d}{dt}\right) x = P_y\left(\frac{d}{dt}\right)\left(Q\left(\frac{d}{dt}\right) x\right). \tag{6}$$

Comparing the left hand and right hand side shows that $y_p := Q\left(\frac{d}{dt}\right) x$ is a particular solution of the ODE, which is polynomial-time computable by the structural properties of $\mathrm{Pol}([0,1])$ given in Proposition 23 for $x \in \mathrm{Pol}([0,1]) \cap \mathcal{C}^\infty([0,1])$. Since the general solution of the ODE can be decomposed into the homogeneous solution y_h and a particular solution y_p, we get

$$\mathcal{T}_{1,n}(a_0, (b_j)_{j=0}^n)(x) = y_h + y_p. \tag{7}$$

Finally, note that by Proposition 25 the homogeneous solution is given by $y_h(t) = e^{-a_0 t} y(0)$, i.e., y_h is polynomial-time computable via Proposition 23, and thereby $y_h + y_p \in \mathrm{Pol}([0,1])$. □

The second step in the proof of Theorem 21 is to extend the first-order result to arbitrary linear ODEs. Thereby, we rely on a simple observation linking polynomial divisibility with roots.

Lemma 28. *Let F be an algebraically closed field, and let $p, q \in F[X]$ be polynomials. Then we have*

$$p \mid q \iff \mathcal{Z}(p) \subseteq \mathcal{Z}(q).$$

Now, the general case follows by exploiting connections between the roots of the characteristic polynomials and certain polynomial reduction techniques.

Proposition 29. *For any (m,n)-order polynomial-time computable ODE of the form*

$$\sum_{i=0}^m a_i y^{(i)} = \sum_{j=0}^n b_j x^{(j)},$$

there exists $x \in \mathcal{C}^\infty([0,1]) \cap \mathrm{Pol}([0,1])$ such that $\mathcal{T}_{m,n}((a_i)_{i=0}^m, (b_j)_{j=0}^n)(x)$ is $\#P$-hard if and only if $\mathcal{Z}(P_y) \nsubseteq \mathcal{Z}(P_x)$.

Proof. First, observe that for $x \in \mathrm{Pol}([0,1]) \cap \mathcal{C}^\infty([0,1])$ if $P_y \mid P_x$, then the corresponding output of the ODE is polynomial-time computable by the same argument as in the proof of Proposition 27, in particular (6) and (7). Hence, by Lemma 28 we know that $\mathcal{Z}(P_y) \subset \mathcal{Z}(P_x)$ implies that for $x \in \mathrm{Pol}([0,1]) \cap \mathcal{C}^\infty([0,1])$ the corresponding solution $y = \mathcal{T}_{m,n}((a_i)_{i=0}^m, (b_j)_{j=0}^n)(x) \in \mathrm{Pol}([0,1])$ is polynomial-time computable.

Thus, assume that $\mathcal{Z}(P_y) \nsubseteq \mathcal{Z}(P_x)$, i.e., there exists $\tilde{\sigma} \in \mathcal{Z}(P_y) \setminus \mathcal{Z}(P_x)$. We show that there exists $x \in \mathcal{C}^\infty([0,1]) \cap \mathrm{Pol}([0,1])$ such that $\mathcal{T}_{m,n}((a_i)_{i=0}^m, (b_j)_{j=0}^n)(x)$ is $\#P$-hard by induction on m. The base-case $(1,n)$ was proven in Proposition 27. Therefore, assume that the claim holds for all polynomial-time computable ODEs of order (k,n) with $k < m$ and consider a polynomial-time computable ODE of order (m,n)

$$\sum_{i=0}^m a_i y^{(i)} = \sum_{j=0}^n c_j x^{(j)}, \quad a_i, c_i \in \mathbb{C}_p.$$

We now make the following observation: If there exist $b_1, \ldots, b_{m-1}, \sigma \in \mathbb{C}_p$ such that

$$\sum_{i=0}^m a_i y^{(i)} = \sum_{i=0}^{m-1} b_i (y' - \sigma y)^{(i)}, \tag{8}$$

then $\tilde{y} := y' - \sigma y$ is a solution of the polynomial-time computable ODE

$$\sum_{i=0}^{m-1} b_i \tilde{y}^{(i)} = \sum_{j=0}^n c_j x^{(j)}. \tag{9}$$

Hence, by the induction hypothesis we find $x \in \mathrm{Pol}([0,1]) \cap \mathcal{C}^\infty([0,1])$ such that $\tilde{y} = \mathcal{T}_{m-1,n}((b_i)_{i=0}^{m-1}, (c_j)_{j=0}^n)(x)$ is $\#P$-hard (which implies that y is also $\#$P-hard by Proposition 23) provided that $\mathcal{Z}(\tilde{P}_y) \nsubseteq \mathcal{Z}(P_x)$ with $\tilde{P}_y(X) := \sum_{i=0}^{m-1} b_i X^i$ being the characteristic polynomial of the left hand side corresponding to the modified ODE (9).

It is left to prove the existence of $b_1, \ldots, b_{m-1}, \sigma$ with the given properties and the fact that $\mathcal{Z}(\tilde{P}_y) \nsubseteq \mathcal{Z}(P_x)$ (or equivalently $\tilde{P}_y \nmid P_x$) holds. Unfolding the condition in (8) gives

$$a_m y^{(m)} + \sum_{i=1}^{m-1} a_i y^{(i)} + a_0 y = b_{m-1} y^{(m)} + \sum_{i=1}^{m-1} (b_{i-1} - \sigma b_i) y^{(i)} - \sigma b_0 y.$$

Hence, comparing the coefficients yields

$$a_m = b_{m-1}, \quad b_{i-1} = a_i + \sigma b_i \text{ for } i = 1, \ldots, m-1, \quad \text{and} \quad \sigma b_0 = -a_0.$$

Recalling the definition of polynomial reduction in (5) and its properties in Proposition 24, we notice that the condition is satisfied for $\tilde{P}_y = \mathcal{R}_\sigma(P_y)$ where $\sigma \in \mathcal{Z}(P_y)$ and additionally $b_i, \sigma \in \mathbb{C}_p$ since $\mathbb{C}_p$ is an algebraically closed field. Since $|\mathcal{Z}(P_y)| = \deg(P_y) = m \geq 2$ we can in particular choose $\sigma \neq \tilde{\sigma}$ so that again by Proposition 24

$$\mathcal{Z}(\tilde{P}_y) = \mathcal{Z}(\mathcal{R}_\sigma(P_y)) = \mathcal{Z}(P_y) \setminus \{\sigma\},$$

i.e., $\tilde{\sigma} \in \mathcal{Z}(\tilde{P}_y) \setminus \mathcal{Z}(P_x)$ or in other words $\mathcal{Z}(\tilde{P}_y) \nsubseteq \mathcal{Z}(P_x)$. □

Finally, Theorem 21 is now a direct consequence of Proposition 29.

Proof of Theorem 21. Proposition 29 (together with Lemma 28) characterizes the complexity of the solution of a polynomial-time computable linear ODE. In particular, there exists $x \in \mathrm{Pol}([0,1]) \cap \mathcal{C}^\infty([0,1])$ such that the corresponding solution is #P-hard if and only if the characteristic polynomials of the ODE satisfy $P_y \nmid P_x$. Thus, assuming $FP \neq \#P$, the solution is not polynomial-time computable, i.e., the ODE exhibits complexity blowup. □

Proof of Theorem 9 Observe that we can rewrite the LIF model from (1) as

$$A_1 y' + A_0 y = B_0 x,$$

with

$$x = \begin{pmatrix} 0 \\ I_e \end{pmatrix}, \qquad y = \begin{pmatrix} u \\ I \end{pmatrix},$$
$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad A_0 = \begin{pmatrix} \tau_m^{-1} & -\tau_m^{-1} \\ 0 & \tau_s^{-1} \end{pmatrix},$$
$$B_0 = \begin{pmatrix} \tau_s^{-1} & 0 \\ 0 & \tau_s^{-1}, \end{pmatrix},$$

where we neglected constants terms (i.e., u_{rest}) without loss of generality. Now, the statement immediately follows from verifying the conditions in Corollary 22.