# P-SpikeSSM: Harnessing Probabilistic Spiking State Space Models for Long-Range Dependency Tasks

**Malyaban Bal**
The Pennsylvania State University
University Park, PA 16802
mjb7906@psu.edu

**Abhronil Sengupta**
The Pennsylvania State University
University Park, PA 16802
sengupta@psu.edu

## Abstract

Spiking neural networks (SNNs) are posited as a computationally efficient biologically plausible alternative to conventional neural architectures, with their core computational framework primarily using the leaky integrate-and-fire (LIF) neuron model. However, the limited hidden state representation of LIF neurons, characterized by a scalar membrane potential, and sequential spike generation process, poses challenges for effectively developing scalable spiking models to address long-range dependencies in sequence learning tasks. In this study, we develop a computationally efficient scalable probabilistic spiking learning framework for long-range dependency tasks leveraging the fundamentals of state space models. Unlike LIF neurons that rely on the determinitic Heaviside function for a sequential process of spike generation, we introduce a SpikeSampler layer that samples spikes stochastically based on an SSM-based neuronal model while allowing parallel computations. To address non-differentiability of the spiking operation and enable effective training, we also propose a surrogate function tailored for the stochastic nature of the SpikeSampler layer. To enhance inter-neuron communication, we introduce the SpikeMixer block, which integrates spikes from neuron populations in each layer. This is followed by a ClampFuse layer, incorporating a residual connection to capture complex dependencies, enabling scalability of the model. Our models attain state-of-the-art performance among SNN models across diverse long-range dependency tasks of the Long Range Arena benchmark and demonstrate sparse spiking pattern highlighting its computational efficiency.

## 1 Introduction

Spiking neural networks (SNNs) ([1]) have garnered attention as a bio-plausible substitute for traditional artificial neural networks (ANNs). Their appeal stems from their utilization of spike-based communication among neurons, a feature that closely mimics biological processes. In the progression of SNN-based architecture advancements, research has predominantly focused on employing leaky-integrate-and-fire (LIF) neurons [2]. While the dynamics modeled by LIF neurons are deemed biologically plausible, the actual operations within the brain entail additional layers of complexity ([3]) and stochasticity [4] that are not fully captured by the simplified LIF neuron model. Moreover, the sequential state updates and spike generation using a deterministic Heaviside function complicate the training of LIF-based SNN architectures, often requiring computationally expensive methods like backpropagation through time (BPTT) [5]. This fundamental challenge has significantly limited the adoption of SNN models, particularly for complicated sequence learning tasks involving long-range dependencies. In this paper, we move beyond traditional LIF-based spike generation models to develop a computationally efficient probabilistic SNN architecture, designed to effectively tackle long-range dependency tasks that have remained largely under-explored in the spiking domain.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

State Space Models (SSMs) have been recently employed in non-spiking architectures to effectively model sequence learning tasks ([6, 7]). In this work, we employ SSMs to capture temporal dependencies within sequences of input spikes, rather than conventional real-valued data. This approach not only enables computational efficiency but also underscores the remarkable capability of SSMs in analyzing long-term temporal dependencies in spike-based data.

**Probabilistic Spiking State-Space Model:** In this paper, we propose an SNN architecture grounded in a probabilistic state-space neuronal model, which we call **P-SpikeSSM**. We conceptualize the $n$-dimensional hidden state of the underlying SSM as the membrane potential, providing richer representations when compared to the scalar hidden state of an LIF neuron. Dynamics of each neuron is governed by an independent set of parameters, allowing the model to flexibly learn diverse temporal dependencies across neurons, thus enhancing its processing capacity. As outlined in the methodology, instead of real-valued inputs, we feed sequence of spikes into the P-SpikeSSM neuronal model. This enables developing a computationally efficient framework by applying convolution over the sparse spikes, instead of real-valued data. The **SpikeSampler** layer samples spikes from each StochSpikeSSM neurons, enabling parallel operation with minimal overhead.

**Scalable Architecture with SpikeSampler and SpikeMixer:** Although individual P-SpikeSSM neurons can process one input spike sequence, addressing tasks with complex long-range dependencies demands a deeper, more scalable architecture capable of capturing diverse dependencies. To address this, we introduce a robust architecture (Fig. 1) and training framework. We encode the real-valued input sequence, associated with a sequence learning task, into "N" distinct spike trains, which are fed to a layer consisting of "N" corresponding neurons. Each neuron generates spikes stochastically based on its individual input spike train, while the collective activity of the neuron population allows for effectively capturing a diverse range of dependencies across the different input spike sequences. The output spikes from each neuron population in a layer are processed through a **SpikeMixer** layer, facilitating inter-neuron communication. Next, a **FuseClamp** layer performs further aggregation and computes the probability necessary for generating the subsequent spike sequences, which are then passed to the next layer of P-SpikeSSM based neuronal units. Furthermore, because the model communicates and uses sparse spike trains for computation, it achieves substantial computational efficiency by significantly reducing the number of floating-point multiplication and accumulation (MAC) operations across all layers and using simpler accumulative operations instead.

**Application to Long-Range Dependency Tasks:** We evaluate the performance and computational efficiency of our proposed spiking architecture on various datasets within the Long Range Arena (LRA) benchmark. Our model outperforms traditional non-spiking transformer-based architectures and, to the best of our knowledge, establishes a new benchmark for fully spiking models in the domain of long-range arena tasks.

## 2 Methodology

In this section, we first delve into the dynamics of the proposed Probabilistic Spiking State Space Models (P-SpikeSSM). We then delve into the specifics of our proposed spiking architecture, highlighting the SpikeSampler, SpikeMixer, and FuseClamp layers. Additionally, we offer insights on scaling the P-SpikeSSM-based spiking model for tackling complex long-range dependency tasks and develop a computationally efficient parallel training framework.

### 2.1 P-SpikeSSM Formulation

We formulate the neuronal model as a time invariant system which takes in sequence of input spikes given as $x_s(t) \in \{0, 1\}$, at time $t$. Much like the membrane potential upholds the state of the LIF neuron, we anchor our approach in SSMs [7, 6], crafting an $n$-dimensional hidden state ($h(t) \in \mathbb{R}^n$) at time $t$. Expanding the dimensionality of the hidden state enables our neuronal model to achieve a more comprehensive state encoding of the underlying input sequence, surpassing the limitations imposed by the scalar hidden state in LIF models. The event of spike generation at time $t$ is associated with a Bernoulli random variable $S_t$ corresponding to each neuron. The probability of spiking, i.e., $p_s(t)$ at time $t$, is modelled as a function of the output of the neural model. The continuous time neuronal dynamics are expressed as,

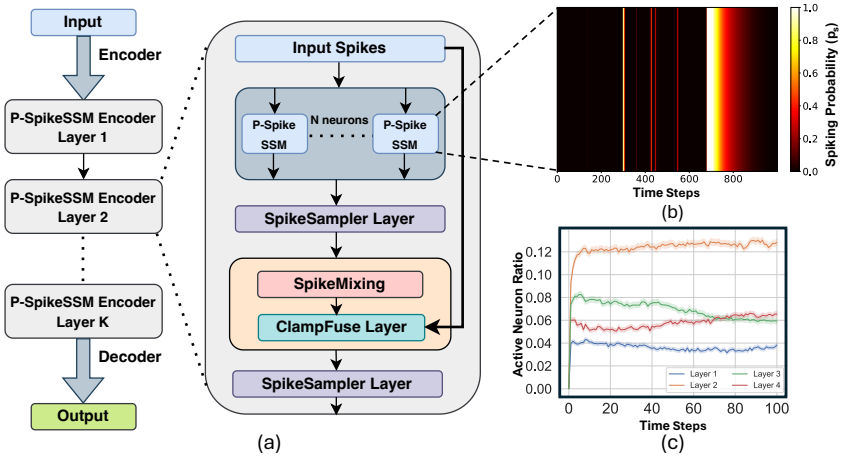


Figure 1: (a) High-level overview of the P-SpikeSSM-based spiking architecture for LRA tasks. (b) Heatmap depicting the sparsity of spiking events generated by a single P-SpikeSSM neuron over input sequence length. (c) Graph showing the layer-wise active neuron ratio (i.e., the proportion of neurons generating spikes within a layer per time step) against operating time steps. The layer-wise spiking behavior illustrates the model-wide sparsity in spiking activity, contributing to computational efficiency.

$$\begin{aligned} \dot{h}(t) &= Ah(t) + Bx_s(t) \\ p_s(t) &= \sigma(Ch(t) + Dx_s(t)) \\ \sigma(z) &= clamp(az + b) \end{aligned} \tag{1}$$

where, $\dot{h}(t) = \frac{dh}{dt}$ and $clamp(y) = \begin{cases} 0 & \text{if } y < 0 \\ y & \text{if } 0 \leq y \leq 1 \\ 1 & \text{if } y > 1 \end{cases}$, $a$ and $b$ are parameters. Setting $a = 1$ and $b = 0$ allows using the output of the SSM directly as the probability of spiking event without further scaling or translation. $A$ is a parameter controlling the evolution of the hidden state over time without any input spikes. $B$ represents the influence of the input spikes ($x_s(t)$). $C$ describes the mapping of the hidden state vector $h(t)$ to the observed outputs, i.e., $p_s(t)$. $D$ is the feedforward parameter, representing any direct influence of the inputs spikes $x_s(t)$ on the observed output probability $p_s(t)$. For the purpose of simpler formulation, following previous works on SSMs [7], we will consider $D = 0$, since the term $Dx_s(t)$ can be viewed as a simple skip-connection. Furthermore, $\sigma$ is a function that clamps the output between $[0, 1]$, since probability $p_s[t] \in [0, 1]$. We utilize $p_s[t]$ to sample spikes from the underlying neuron, as discussed in Section 2.1.3. The aforementioned formulation of our SSM-based neuronal model is presented in a continuous-time setting. However, since our primary focus is on sequence modeling tasks in domains such as NLP and vision tasks, we now proceed to formulate the dynamics of our neuronal model in discrete time.

### 2.1.1 P-SpikeSSM Discrete Time Dynamics

In order to discretize our system we sample a sequence of spikes of length $L$, given by $X_s = (x_s[1], x_s[2], ..., x_s[L])$ from the original continuous signal given by $x_s(t)$, with step size $\Delta$ such as $x_s[i] = x_s(i\Delta)$. The P-SpikeSSM neuronal dynamics are subsequently discretized using bilinear transformations [8], whereby we approximate the parameters $A, B, C$ as $\overline{A}, \overline{B}, \overline{C}$ which is given as$\overline{A} = (I - \Delta/2 \cdot A)^{-1}(I + \Delta/2 \cdot A), \overline{B} = (I - \Delta/2 \cdot A)^{-1}\Delta B, \overline{C} = C$,

where, $I$ is the Identity matrix. The transition dynamics of the discretized system at time step $t$ is,

$$\begin{aligned} h[t] &= \overline{A}h[t-1] + \overline{B}x_s[t] \\ p_s[t] &= \sigma(\overline{C}h[t]) \end{aligned} \tag{2}$$

where, $h[t]$ is the hidden state of the neuron, $p_s[t]$ is the probability of the event $S_t$. This allows us to write the transition dynamics of the system as a recurrence in discrete time. The sparse spiking dynamics of our proposed neuronal model, characterized by the spike probability $p_s[t]$, is illustrated in Fig. 1b. The spiking activity manifests in temporally localized patterns, mirroring the firing patterns observed in biological neurons [9], with periods of non-activity interspersed between bursts. Now, instead of using a recurrent representation, we investigate how the evolution of state dynamics can be represented by a convolution operation (with spikes as input signal), thus requiring only accumulation-based computationally efficient operation. Moreover, using convolution instead of a recurrence-based approach enables us to parallelize the framework.

### 2.1.2 Representing Dynamics as Convolution over Spikes

There are two problems with Eqn. 2, concerning the training of a scalable spiking architecture. Firstly, training it in its recurrent form necessitates employing a BPTT approach ([10]), which is impractical for longer sequence lengths due to its time and memory overhead. Secondly, as the hidden state $h[t]$ at time $t$ can be a vector of floating points rather than spikes, the transition operations involved would not take complete advantage of energy/power efficient neuromorphic hardware. To achieve a fully parallelizable training procedure and leverage SNN-based operational efficiency during inference, we investigate an alternative formulation of Eqn.2 as a convolution operation [10]. Since the proposed neuronal model is a time invariant system, considering the initial hidden state i.e., $h[0]$ to be a 0-vector, the recurrent relationship can be unrolled as,

$$h[i] = \overline{A}^{i-1}\overline{B}x_s[1] + \overline{A}^{i-2}\overline{B}x_s[2] + \cdots + \overline{AB}x_s[i-1] + \overline{B}x_s[i] = \sum_{j=1}^{i}(\overline{A}^{i-j}\overline{B}x_s[j]) \quad (3)$$

Thus, generalizing it to the entire sequence of length $L$ we get, $H = \hat{K} * X_s$, $\hat{K} = (\overline{B}, \overline{AB}, \ldots, \overline{A}^{L-1}\overline{B})$ where, $*$ represents the non-circular convolution operation. $H$ represents the sequence of hidden states $(h[1], h[2], \ldots, h[L])$ of length $L$. $\hat{K}$ is a convolutional kernel of length $L$ as defined above. The output of the neuronal model, i.e. probability of spiking of the neuron at time $t$, is given as,

$$p_s[i] = \sigma((K * X_s)_i) \quad (4)$$

where, kernel $K = \overline{C}\hat{K} = (\overline{CB}, \overline{CAB}, \ldots, \overline{CA}^{L-1}\overline{B})$; $(K * X_s)_i = \sum_{j=1}^{i} K_j x_s[i-j+1]$ is the $i^{th}$ term of the non-circular convolution, where $K_i = \overline{CA}^{i-1}\overline{B}$. $P_s = (p_s[1], p_s[2], \ldots, p_s[L])$ is the sequence of probability of spikes from a neuron over the operating time steps. Thus, we can compute the output sequence parallely by doing convolution of the input sequence of spikes with the weights of kernel $K$. The demonstrated sparsity of spikes, as depicted in Fig. 1c, enables leveraging more efficient fast-fourier transform (FFT) implementations, such as Sparse FFT [11] during traning. Additionally, each element of the sequence $P_s$ can be computed as a dot product of a vector of real values (elements of precomputed $K$) and vector of spikes (subsequence of $X_s$). Sparse input spikes further enables skipping unnecessary computations on zero elements within the input signal. Specialized neuromorphic hardware accelerators [12, 13] can be leveraged to perform this process, thus avoiding floating point MAC operations during inference.

### 2.1.3 SpikeSampler Layer

The spiking event of a specific P-SpikeSSM neuron at time $t$ is modeled by a Bernoulli random variable $S_t$. The spike generation process utilizes the output of the neuron, $p_s[t]$ (Eqn. 2 & 4), as the probability of spiking at time $t$, as demonstrated below:

$$S_t = \begin{cases} 1 & \text{if } z < p_s[t], \\ 0 & \text{otherwise,} \end{cases} \qquad z \sim \mathcal{U}(0,1) \quad (5)$$

This seemingly simple sampling process allows the proposed framework to operate in parallel without incurring significant computational overhead. This SpikeSampling process can be parallelized across both the sequence dimension and the population of neurons $N$, to develop a SpikeSampler layer (Fig. 1a), facilitating the scaling of this methodology for more complex long-range dependency tasks.

**Surrogate Gradient:** The stochastic nature of the SpikeSampler layer introduces challenges during training, as P-SpikeSSM-based spiking architectures face the problem of non-differentiability of spikes. To address this and enhance stability of the learning process, during the backward phase of backpropagation, we use a surrogate operation ($\overline{S}_t$) for the stochastic spiking operation at time $t$ as,

$$\overline{S}_t = \mathbb{E}[S_t] = 0 \cdot P(S_t = 0) + 1 \cdot P(S_t = 1) = p_s[t] \tag{6}$$

**Parallel Execution in Our Model:** The SpikeSampler layer compliments the parallel computational advantages of the P-SpikeSSM neuron (Eqn. 4), resulting in a parallel and efficient spike generation process without the additional overhead of an LIF neuron). During the backward phase, the surrogate gradient (Eqn. 6) is utilized for effective and efficient model training. To efficiently compute the kernel $K$, we capitalize on prior theoretical structural findings regarding non-spiking SSMs [6]. By exploiting the decomposition of matrix $A$ into a sum comprising a normal and low-rank matrix, we achieve efficient computation of $K$ in $O(L)$ time complexity. More details regarding efficient computation of $K$ is added in the Appendix B.

### 2.2 Scaling P-SpikeSSMs to Deeper SNN Architectures

To expand the sequence learning capabilities of the P-SpikeSSM neuronal model and facilitate its scalability to deeper architectures, we introduce the P-SpikeSSM neuronal layer. This layer comprises of $N$ P-SpikeSSM neurons. The sparse spiking activity at layer $i$ at time $t$ can be characterized by analyzing the active neuron ratio, denoted as $anr_i[t]$. This ratio is determined by the number of spikes occurring in that layer at time $t$, and is defined as: $anr_i[t] = \frac{\sum_{j=1}^{N_i} s_{ij}[t]}{N_i}$, where $N_i$ is the total number of neurons in layer $i$, and $s_{ij}[t]$ represents the spiking state of neuron $j$ at time $t$. The sparse spiking activity of the P-SpikeSSM neurons is illustrated in Fig. 1c. The high-level architecture of the SNN features $M$ stacked P-SpikeSSM Encoder layers, each of which encapsulates the layers as shown in Fig. 1a.

**SpikeMixer Block:** Since each P-SpikeSSM neuron independently processes input sequence tokens, a neuron mixer layer in the form of a fully-connected block is introduced. This facilitates the aggregation of spikes from previous layer of P-SpikeSSM neurons and efficient flow of information among neuronal layers, ensuring efficient processing of diverse temporal dependencies learned by various neurons. The output of the SpikeMixer operation is given as $f_{mix}[t] = gelu(I_s[t] \cdot W_{fc}) where$,$\mathbf{I_s}[t] \in \{0,1\}^N$ are the $N$ spikes from the previous SpikeSampler layer at time $t$ and $W_{fc} \in \mathbb{R}^{N \times N}$ is a linear weight. We use $gelu$ function as a non-linearity. The linear layer in this module avoids floating-point MAC operations since the input to the FC block consists of spikes. However, due to the $gelu()$ activation, there is element-wise floating-point multiplication of $O(n^2)$ complexity [14], which is still lower than the $O(n^3)$ MAC operation in floating-point matrix multiplications.

**FuseClamp Block:** The FuseClamp block contains a neuronal layer that combines input spikes with the SpikeMixer output via a residual connection. This is followed by normalization ($N$), which after which the output is clamped between $[0, 1]$. As in the P-SpikeSSM neuronal layer, the output is interpreted as the probability for spiking event at time $t$ for the FuseClamp layer. The operation is defined as follows: $p_{cf_i}[t] = \sigma(N(f_{mix}[t] + x_s[t]))$

## 3 Experimentation

In this section, we showcase the efficacy of our proposed S6 neuronal model-based SNN architectures by evaluating their performance across various long-range dependency based tasks. Additionally, we conduct analyses to assess net spiking activity and energy/power efficiency of our model. The experiments were run on Nvidia RTX A5000 GPUs (8) each with 24GB memory.

Table 1: Results showing performance of our model against some spiking and non-spiking architectures on test sets of LRA benchmark tasks. Accuracy is used as the metric for all the tasks.

| Model | SNN | ListOps | Text | Retrieval | Image | Pathfinder |
|---|---|---|---|---|---|---|
| S4 (Original) [6] | No | 58.35 | 76.02 | 87.09 | 87.26 | 86.05 |
| S4 (Improved) [6] | No | 59.60 | 86.82 | 90.90 | 88.65 | 94.20 |
| Transformer [15] | No | 36.37 | 64.27 | 57.46 | 42.44 | 71.40 |
| Sparse Transformer[16] | No | 17.07 | 63.58 | 59.59 | 44.24 | 71.71 |
| Linformer [17] | No | 35.70 | 53.94 | 52.27 | 38.56 | 76.34 |
| Linear Transformer ([16]) | No | 16.13 | 65.90 | 53.09 | 42.34 | 75.30 |
| FLASH-quad [18] | No | 42.20 | 64.10 | 83.00 | 48.30 | 63.28 |
| Spiking LMUFormer [19] | Yes | 37.30 | 65.80 | 79.76 | 55.65 | 72.68 |
| Transnormer T2 [20] | No | 41.60 | 72.20 | 83.82 | 49.60 | 76.80 |
| BinaryS4D [21] | Partial | 54.80 | 82.50 | 85.30 | 82.00 | 82.60 |
| **P-SpikeSSM (Our Model)** | **Yes** | **58.20** | **81.20** | **88.53** | **80.60** | **84.80** |

### 3.1 Long Range Arena Benchmark

To demonstrate the long-range dependency analysis capability of our S6 based architecture, we leverage the Long Range Arena (LRA) benchmark [16]. This benchmark spans various classification tasks from textual to image domains. The five tasks in LRA are explained further in the appendix.

While the tasks in the LRA benchmark are highly relevant from a neuromorphic perspective—such as processing byte sequences for NLP and pixel sequences for image tasks—this area remains largely unexplored in neuromorphic computing. This is primarily due to two reasons: the inherent limitation in information retention capacity of vanilla LIF-based SNNs, and the scalability challenge encountered when training SNNs using BPTT (akin to an RNN) on lengthy sequence lengths. The latter issue arises from the increased memory overhead resulting from the computational graph. Moreover, transformer-based non-spiking architectures, which we use for comparison, exhibit suboptimal performance due to the overhead (while computing attention scores) associated with longer sequence lengths. In the spiking domain, we contrast our results with those of the spiking version of LMU.

#### 3.1.1 Ablation Study

We perform experiments to analyze the effect of various components introduced in this work, specifically the surrogate (used during training) for the SpikeSampler layer, the SpikeMixer, and use of normalization in ClampFuse layer. The results are shown in Table 2. An energy-analysis is done in Appendix F to highlight the computational advantage of our method.

| Our Model | Accuracy |
|---|---|
| w/o Surrogate ($S_t$) | 70.90 |
| w/o SpikeMixer | 68.90 |
| w/o Normalization | 77.80 |
| w/ All Components | 81.20 |

Table 2: Results showing the effect of different components of our proposed SNN architecture on test accuracy when trained on LRA Text dataset.

## 4 Conclusions

We propose a computationally efficient probabilistic spiking framework for addressing long-term dependency sequence learning tasks. Intead of using LIF neurons, our model uses the output of P-SpikeSSM neuronal model as the probability for generating spike using the proposed SpikeSampler layer. To tackle the non-differentiability of this stochastic spiking mechanism, we introduce a surrogate gradient approach, enabling efficient training. To ensure scalability our architecture features SpikeMixer and ClampFuse layers, enabling effective sequence processing through simplified accumulation-based operations. We evaluate our models on multiple tasks of the LRA benchmark. Our models consistently outperform transformer-based non-spiking counterparts, achieving state-of-the-art performance among SNN models, while also demonstrating exceptional computational efficiency due to the inherent sparsity of spiking events.

**Limitations and Future Works:** To fully capitalize on the energy and power efficiency advantages, future next steps can consider deploying this model on edge-based devices and neuromorphic chips, such as the Intel Loihi 2.

## Acknowledgments

This material is based upon work supported in part by the U.S. National Science Foundation under award No. CCSS #2333881, CAREER #2337646, CCF #1955815, and EFRI BRAID #2318101.

## References

[1] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *International journal of neural systems*, 19(04):295–308, 2009.

[2] Anthony N Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95:1–19, 2006.

[3] AL Hodgkin and AF Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Bulletin of mathematical biology*, 52:25–71, 1990.

[4] LM Harrison, O David, and KJ Friston. Stochastic models of neuronal dynamics. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1457):1075–1091, 2005.

[5] 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.

[6] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[7] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[8] Arnold Tustin. A method of analysing the behaviour of linear systems in terms of time series. *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms*, 94(1):130–142, 1947.

[9] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[10] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021.

[11] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Simple and practical algorithm for sparse fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1183–1194. SIAM, 2012.

[12] Dmitry Ivanov, Aleksandr Chezhegov, and Denis Larionov. Neuromorphic artificial intelligence systems. *Frontiers in Neuroscience*, 16:959626, 2022.

[13] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

[14] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[16] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

[17] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[18] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International conference on machine learning*, pages 9099–9117. PMLR, 2022.

[19] Zeyu Liu, Gourav Datta, Anni Li, and Peter Anthony Beerel. Lmuformer: Low complexity yet powerful spiking model with legendre memory units. *arXiv preprint arXiv:2402.04882*, 2024.

[20] Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear transformer. *arXiv preprint arXiv:2210.10340*, 2022.

[21] Matei Ioan Stan and Oliver Rhodes. Learning long sequences in spiking neural networks. *arXiv preprint arXiv:2401.00955*, 2023.

[22] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

[23] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[24] Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer. *arXiv preprint arXiv:2209.15425*, 2022.

[25] Malyaban Bal and Abhronil Sengupta. Spikingbert: Distilling bert to train spiking language models using implicit differentiation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10998–11006, 2024.

[26] Malyaban Bal, Yi Jiang, and Abhronil Sengupta. Exploring extreme quantization in spiking language models. *arXiv preprint arXiv:2405.02543*, 2024.

[27] Rui-Jie Zhu, Qihang Zhao, and Jason K Eshraghian. Spikegpt: Generative pre-trained language model with spiking neural networks. *arXiv preprint arXiv:2302.13939*, 2023.

[28] Malyaban Bal and Abhronil Sengupta. Equilibrium-based learning dynamics in spiking architectures. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2024.

[29] Aaron R Voelker, Daniel Rasmussen, and Chris Eliasmith. A spike in performance: Training hybrid-spiking neural networks with quantized activation functions. *arXiv preprint arXiv:2002.03553*, 2020.

[30] Yu Du, Xu Liu, and Yansong Chua. Spiking structured state space model for monaural speech enhancement. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 766–770. IEEE, 2024.

[31] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[32] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015.

## A Deriving Convolutional Representation of S6 Neuronal Dynamics

At time step $t$, the discretized S6 neuronal model exhibits transition dynamics described as follows:

$$\begin{aligned} h[t] &= \overline{A}h[t-1] + \overline{B}x_s[t] \\ p_s[t] &= \sigma(\overline{C}h[t]) \end{aligned} \tag{S1}$$

where, $h[t]$ is the hidden state of the neuron, $p_s[t]$ is the probability of the spiking event $S_t$. $\overline{A}$, $\overline{B}$, $\overline{C}$ are the discretized parameters of the time-invariant system. Considering at time step 0, $h[0] = 0$, we get,

$$\begin{aligned} h[1] &= \overline{B}x_s[1] \\ h[2] &= \overline{AB}x_s[1] + \overline{B}x_s[2] \end{aligned} \tag{S2}$$

Unrolling like this to time step $i$ we get,

$$h[i] = \overline{A}^{i-1}\overline{B}x_s[1] + \overline{A}^{i-2}\overline{B}x_s[2] + \cdots + \overline{AB}x_s[i-1] + \overline{B}x_s[i] = \sum_{j=1}^{i}(\overline{A}^{i-j}\overline{B}x_s[j]) \tag{S3}$$

Thus, the convolutional kernel $\hat{K}$, whose length is given by the length of the input sequence $L$, is defined as,

$$\hat{K} = (\overline{B}, \overline{AB}, \ldots, \overline{A}^{L-1}\overline{B}) \tag{S4}$$

Now $H$, i.e., sequence of hidden states can be computed as a non-circular convolution given as, $H = \hat{K} * X_s$, where $X_s$ is the input sequence of spikes.

Thus, $p_s[t] = \sigma((K * X_s)_t) = \sigma(\sum_{j=1}^{t} K_j x_s[t-j+1])$, where $K = (\overline{CB}, \overline{CAB}, \ldots, \overline{CA}^{L-1}\overline{B})$.

## B HiPPO-legS Matrix

HiPPO (high-order polynomial projection operators) [22] is a versatile framework that enables the analysis of various families of measures. Utilizing this operator as either a closed-form ordinary differential equation (ODE) or a linear recurrence, we can efficiently update the optimal polynomial approximation as the input function unfolds over time. HiPPO-legS can generalize to different time scales. HiPPO enables the hidden state to effectively memorize the historical pattern of input spikes (in our paper). The elements of the HiPPO-legS (Scaled Legendre) matrix $\in \mathbb{R}^{n \times n}$ is given below,

$$A_{mk} = -\begin{cases} \sqrt{2m+1}\sqrt{2k+1}, & \text{if } m > k \\ m+1, & \text{if } m = k \\ 0, & \text{if } m < k \end{cases} \tag{S5}$$

### B.1 Computing Kernel $K$

The efficient computation of $K$ has been proposed in the literature [6], thus speeding up the parallel training of the SSM based neuronal architectures. We briefly go over the overview on how it is achieved. The primary concern in computing $K$ is the repeated multiplication of the state matrix to create the individual terms $K_i$. Thus to compute $K$, the time complexity for a simple approach of chained multiplication is $O(n^2L)$, were $n$ is the hidden state dimension and $L$ is the sequence length. Now the idea is that, if we had the state matrix to be a diagonal matrix, then theoretically we could compute $K$ efficiently using Vandermonde product. Thus, the goal is to diagonalize the matrix $A$. Now, the ideal scenario is if $A$ is a normal matrix, i.e., it is diagonalizable with a unitary matrix ($UAU^{-1}$ is a diagonal matrix, where $U$ is a square matrix such that $U^H = U^{-1}$). $A$ is initialized to HiPPO matrices which are not normal matrices. However, HiPPO can be decomposed into a diagonal matrix ($\Lambda$) and a low-rank matrix. Following this, we can leverage previous theoretical results [6] on

reducing the underlying SSM to the computation of Cauchy kernels and calculate $K$, in linear order of time complexity w.r.t the sequence length $L$.

## C Related Works

The realm of sequence modeling is primarily dominated by transformer-based architectures. Efficient implementations like LinFormer ([17]), Performer ([23]), among others, have demonstrated scalability to long sequence lengths. Meanwhile, non-spiking architectures based on SSMs, such as S4 and Mamba ([7, 6, 22]), have also shown the capability to handle lengthy sequences. Sequence learning in SNN-based architectures have primarily been applied to vision-based datasets ([24]) and NLP datasets ([25, 26, 27, 28]), typically with constrained sequence lengths. However, within the domain of SNNs, frameworks based on Legendre Memory Units (LMUs) ([19, 29]) has ventured into exploring long-range dependency tasks.

Previous efforts integrating SSMs within spiking models ([21, 30]) have primarily focused on passing the SSM output through a layer of LIF neurons to generate spikes. Using non-linear LIF neurons negates the parallel training efficiency of SSMs, as LIF neurons process information sequentially, introducing a bottleneck (see Section D). Stan and Rhodes [21] seeks to enhance efficiency by linearizing LIF neurons. However, because the inputs to their SSM model remain real-valued, leading to additional floating-point MAC operations, this approach fails to leverage the energy-saving potential of SNNs. Moreover, the work lacks an analysis of computational efficiency and energy benefits, particularly concerning neuron firing activity, leaving a critical aspect of model efficiency unaddressed. Furthermore, from a sequence processing standpoint, the inherent ability and use of SSMs to capture temporal dependencies renders the addition of LIF neurons superfluous, introducing unnecessary computational overhead without providing any functional improvements beyond enabling spike generation.

## D LIF Neuron Dynamics

The continuos-time internal dynamics of a basic LIF neuron is outlined below,

$$\tau_m \cdot \frac{du}{dt} = -(u(t) - u_{rest}) + R \cdot I(t) \qquad \text{(S6)}$$

where, at time $t$, $u(t) \in \mathbb{R}$ is the membrane potential which can be considered as the hidden state of the system; $I(t)$ is the input current scaled by a constant $R$; $\tau_m$ is the time constant associated with the decay in membrane potential over time; $u_{rest}$ is the resting membrane potential. LIF neurons thus sequentially updates its state ($u$) and uses deterministic heaviside function for spikes generation.

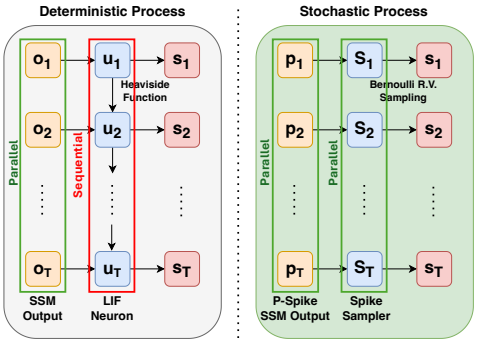


Figure S1: Computational Flow of LIF-based SSM Models and SpikeSampler-driven P-SpikeSSM Neuronal Model.

**Sequential Bottelnecking in LIF neurons:** The above sequential process of state update and spike generation causes a bottleneck during the parallel training of the underlying SSM based framework. This results in increase in both training and inference time (see Section 3.1.1) compared to our method. Prior work [21] attempts to linearize LIF neurons for parallel operation with SSMs, but offers limited analysis on the impact of this parallelization on model performance. In contrast, as shown in our results, our approach not only surpasses the accuracy achieved by previous methods across multiple datasets, but does so by being computationally simpler than the former. Additionally, the previous study has not provided evidence for any contributions of LIF neurons to model performance improvement beyond their use in spike generation.

| Dataset | Accuracy |
|---|---|
| psMNIST | 98.3±0.1 |
| SC10 | 95.4±0.2 |
| LRA: ListOpns | 58.0 ±0.2 |
| LRA: Text | 80.8 ±0.4 |
| LRA: Retrieval | 88.4 ±0.1 |
| LRA: Image | 80.0 ±0.6 |
| LRA: Pathfinder | 84.6 ±0.2 |

Table S1: Accuracy statistics across different datasets.

Table S2: Hyper-parameters used for obtaining the best result on individual datasets used for evaluating S6-based SNN models.

| | psMNIST | SC10 | ListOpns | Text | Retrieval | Image | Pathfinder |
|---|---|---|---|---|---|---|---|
| $M$: #S6 Encoder Layers | 2 | 4 | 4 | 4 | 4 | 4 | 4 |
| $N$: Neurons per Layer | 400 | 256 | 256 | 256 | 200 | 256 | 256 |
| $n$: Hidden State Dim. | 64 | 32 | 32 | 16 | 32 | 32 | 64 |
| $lr$: Learning Rate | 0.01 | 0.002 | 0.005 | 0.0005 | 0.003 | 0.005 | 0.0005 |
| Batch Size | 64 | 32 | 32 | 64 | 32 | 32 | 32 |
| Epochs | 100 | 50 | 50 | 80 | 50 | 150 | 150 |

## E Additional Experimental Results

In Table S2, we list the optimal set of hyper-parameters used for each of the tasks. Primarily batch normalization was used for normalization. Initializing the state matrix $A$ with HiPPO matrices [22] leads to optimal performance and rapid convergence. Across a majority of tasks, utilizing HiPPO-legS (Appendix B) consistently yields the highest accuracy. The step size for discretization ($\Delta$) is restricted between $[0.001, 0.1]$. The comprehensive statistics of accuracy obtained after running 10 different instances of each experiment with random seeds is shown in Table S1.

**Dataset Details:Long Range Arena Benchmark**

To demonstrate the long-range dependency analysis capability of our spiking architecture, we leverage the Long Range Arena (LRA) benchmark [16], spanning various classification tasks from textual to image domains. Following are the five tasks utilized for evaluation,

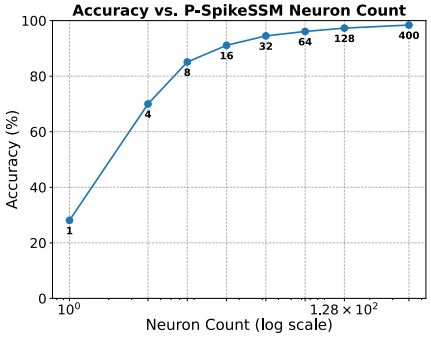


Figure S2: Results obtained from the test set of the ps-MNIST dataset. This experiment utilizes two P-SpikeSSM neuronal layers, with each layer containing $n$ neurons, represented on the x-axis. The accuracy achieved is shown on y-axis.

**ListOps**: In this task, our focus lies in modeling hierarchically structured data within a long-context framework. The sequence length for this task is upto $2K$.

**Text**: In this task, we process the IMDB dataset [31] of movie reviews and perform the task of sentiment analysis in a byte-level. This is done to ensure a long sequence length of $4K$.

**Retrieval**: In this task, we assess the model's capacity to encode and retain compressed representations essential for matching and retrieval purposes. The input consists of byte-level sequences (of length $4K$) from two documents, and the goal is to analyze their similarity.

**Image**: In this task, we perform an image classification task based on a sequence of pixels of the original image. The dataset is CIFAR-10 and the sequence length is $1K$.

**Pathfinder**: In this task, we treat a $32 \times 32$ image as a sequence of pixels of length $1K$. Our objective is to make a binary decision regarding whether two points, depicted as circles, are linked by a path composed of dashes.

**Memory Footprint:** Based on previous analysis [16] on LRA tasks, the Transformer models in Table 1 are configured with $4$ layers, hidden dimension of $256$ and $4$ attention heads, resulting in

approximately $600K$ parameters in total. Our models establish state-of-the-art performance in the spiking domain and comprehensively outperforms non-spiking transformer based architectures as shown in Table 1. Furthermore, considering identical parameters $(\overline{A}, \overline{B}, \overline{C})$ for neurons in the same layer, the average parameter count of our models across all five LRA tasks is around $\approx 250K$, representing a reduction of $\approx 2.4\times$ compared to the parameter count of the transformers used.

**Effect of number of Neurons:** The effect of the number of neurons on model performance is demonstrated in Fig. S2.This demonstrates that as the population of neurons within a single layer increases, the spikes generated by them more effectively capture the temporal dependencies in the input sequence.

## F Analysis of Energy Efficiency

We perform a preliminary analysis comparing the energy efficiency of a non-spiking S4 model to our spiking model during parallel execution on a 45nm CMOS circuit ([32]). For 32-bit floating points, ACC operations (cost $.9pJ$) consume $5.1\times$ less energy than MAC operations (cost $4.6pJ$) ([32]). Assuming an input sequence length of $L$, with $N$ neurons per layer across $K$ layers, the dominant energy cost per layer for the non-spiking S4 model [6] is $(L^2 + LN^2)$ floating point MAC operations, representing the combined cost of computation (underlying SSM is operated parallely) in a single S4 layer and following linear layer.

For simplicity, we estimate the energy costs using standard convolution instead of FFT, as implementing FFT on a neuromorphic chip—which primarily relies on spike-based accumulative operations—is significantly more complex. Although a complete energy calculation includes non-linear layers like $gelu()$ and $Norm$, their contribution is negligible ($O(LN)$ operations) compared to the energy cost of the parallel SSM and linear layers. Building on our previous analysis, the primary computational cost per P-SpikeSSM Encoder Layer in our spiking model is given by $(IFR_{\text{in}} \cdot L^2 + IFR_{\text{o}} \cdot LN^2)$ floating-point accumulation (ACC) operations, contributed primarily by the parallel operation of the P-SpikeSSM and SpikingMixer layer. Here, $IFR_{\text{in}}$ represents the firing rate of the input layer to the P-SpikeSSM neuronal layer, while $IFR_{\text{o}}$ denotes the firing rate of spikes sampled from the P-SpikeSSM neuronal layer, i.e. input to the SpikeMixer. As illustrated in Fig. S3, the majority of neurons in the layer remain dormant during the input sequence, leading to sparse communication.

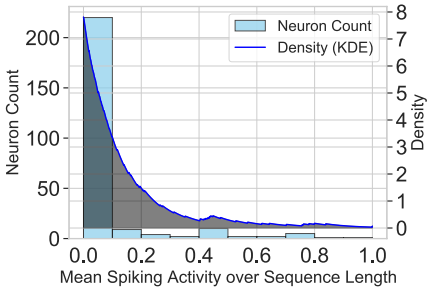


Figure S3: Results obtained after passing randomly sampled inputs from ListOps dataset of LRA benchmark through our model. Figure consists of Histogram representing the count of neurons associated with mean probability of spiking (averaged over the entire sequence of length $L$) and Kernel density estimation (KDE) plot of the data using an exponential kernel.

To illustrate with a specific example, let us consider the ListOps dataset. An iso-parametric state-of-the-art non-spiking S4 model achieves an accuracy of $58.35$ (improved version: $59.60$) ([6]), while our P-SpikeSSM achieves $58.20$. However, the energy consumption of the non-spiking model is $4 \times (2K \times 2K + 2K \times 256 \times 256)$ MAC operations, resulting in a total energy consumption of $2.55$ mJ, whereas our spiking model consumes only $0.0398$ mJ. Consequently, our model is $> 64\times$ more energy efficient based on computational cost. Although this methodology does not include architectural details in the energy analysis, it still highlights the computational efficacy of our approach. By leveraging the prevalence of inactive neurons and sparse spiking patterns of active neurons, we achieve significant improvements in energy and power efficiency on neuromorphic platforms.