Efficient and Effective Time-Series Forecasting with Spiking Neural Networks

Changze Lv [1] **Yansen Wang** [2] **Dongqi Han** [2] **Xiaoqing Zheng** [1] **Xuanjing Huang** [1] **Dongsheng Li** [2]

Abstract

Spiking neural networks (SNNs), inspired by the spiking behavior of biological neurons, provide a unique pathway for capturing the intricacies of temporal data. However, applying SNNs to time-series forecasting is challenging due to difficulties in effective temporal alignment, complexities in encoding processes, and the absence of standardized guidelines for model selection. In this paper, we propose a framework for SNNs in time-series forecasting tasks, leveraging the efficiency of spiking neurons in processing temporal information. Through a series of experiments, we demonstrate that our proposed SNN-based approaches achieve comparable or superior results to traditional time-series forecasting methods on diverse benchmarks with much less energy consumption. Furthermore, we conduct detailed analysis experiments to assess the SNN's capacity to capture temporal dependencies within time-series data, offering valuable insights into its nuanced strengths and effectiveness in modeling the intricate dynamics of temporal data. Our study contributes to the expanding field of SNNs and offers a promising alternative for time-series forecasting tasks, presenting a pathway for the development of more biologically inspired and temporally aware forecasting models. Our code is available at `https://github.com/microsoft/SeqSNN`.

1. Introduction

Spiking neural network (SNN) is regarded as the third generation of neural network (Maass, 1997) for its energy efficiency, event-driven paradigm, and biological plausibility. Nowadays, SNNs have achieved comparable performance with artificial neural networks (ANNs) in image classification (Hu et al., 2018; Fang et al., 2021; Ding et al., 2021; Zhou et al., 2023b; Yao et al., 2023a), text classification (Lv et al., 2023b;a), sequential image classification (Jeffares et al., 2022; Chen et al., 2023), and time-series classification (Dominguez-Morales et al., 2018; Fang et al., 2020a). Nevertheless, the existing studies either completely neglect the temporal nature of SNN or oversimplify the incorporation of data into the event-driven paradigm despite the sequential data format, e.g., by repeating the samples along the time axis (Fang et al., 2021; Lv et al., 2023b) or only preserving changes across data points (Reid et al., 2014; Fang et al., 2020a). These strategies, while serving their purpose, can not fully exploit the advantages of SNNs in the domain of temporal signal processing.

Remarkably, to cater to the event-driven paradigm which SNNs prefer, neuromorphic image datasets such as CIFAR-10-DVS (Li et al., 2017a) and DVS-128-Gesture (Amir et al., 2017) have been created from dynamic vision sensors (DVS) (Leñero-Bardallo et al., 2011). DVS operates in an event-driven manner, only transmitting information when there is a change in the scene (pixel intensity changes), which is well-aligned with the spiking nature of SNNs. SNNs have demonstrated outstanding performance (Zhou et al., 2023b; Yao et al., 2023a) upon these neuromorphic datasets, showing their potential to be not only conceptual for energy efficiency but also powerful and competitive in pursuing state-of-the-art results with built-in temporal information. However, the acquisition of dynamic image datasets for such evaluations is often encumbered by high costs and logistical inconveniences, thereby posing challenges in aligning with the pragmatic requirements of real-world applications.

Acknowledging the mismatch between the preferable data format of SNNs and practical needs, we identify time-series forecasting as the potential ideal task. Time-series forecasting, a vital aspect of realistic data analysis including traffic (Li et al., 2017b), energy (Lai et al., 2018), etc., aims to predict future values based on historical observations arranged chronologically. Addressing this task often involves modeling the temporal dynamics, resonating profoundly with the nature of neural coding.

Although SNNs are effective at managing temporal informa-

The work was conducted during the internship of Changze Lv (czlv22@m.fudan.edu.cn) at Microsoft Research Asia. [1]School of Computer Science, Fudan University, Shanghai, China [2]Microsoft Research Asia, Shanghai, China. Correspondence to: Xiaoqing Zheng <zhengxq@fudan.edu.cn>, Yansen Wang <yansenwang@microsoft.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

tion, applying them to time-series forecasting tasks remains insufficiently explored due to some significant challenges. Firstly, achieving effective temporal alignment between continuous time-series data and the discrete spiking periods of SNNs poses a hurdle, requiring careful consideration of encoding mechanisms. A substantial disparity exists between the discrete characteristics of spike values in SNNs and the floating-point attributes of time-series data, necessitating robust mechanisms to mitigate information loss and noise when converting meaningful floating-point values to spike trains. Moreover, the lack of standardized guidelines for proper model selection further complicates the task, calling for a thorough exploration of SNN architectures and their parameters tailored to the specific characteristics of diverse time-series datasets.

In this paper, we propose a framework for SNNs in time-series forecasting tasks. Firstly, by leveraging the efficiency of spiking neurons in processing time sequential information, we successfully align the time steps between time-series data and SNNs. Secondly, we design two types of encoding layers to transfer continuous time-series data to meaningful spike trains. Finally, we modify three types of ANNs (CNNs, RNNs, and Transformers) to their SNN counterparts with no floating-point multiplication and division, aiming to offer a guideline for proper SNN model selection for time-series forecasting tasks in deep learning age. We conduct a comprehensive evaluation of our proposed SNN models on 4 widely-used time-series forecasting benchmarks and the results show that SNNs achieve comparable or even better results to classic ANNs with much less energy consumption. Furthermore, we conduct analysis experiments to show how SNNs capture temporal dependencies within time-series data and find that SNNs can indeed model the inner dynamics of time-series data. To sum up, our contributions can be summarized as follows:

Framework. We propose a unified framework for SNNs in time-series forecasting tasks, including time-series data encoding, and SNN model architecture, which offers an energy-efficient and biological-plausible alternative for time-series forecasting.

Performance. The presented framework enables the performance of the SNN domain to achieve comparable or even superior to existing classic ANN baselines with much less energy consumption.

Insightful Analysis. To the best of our knowledge, this paper stands among the first to provide a thorough analysis, encompassing both model-level investigations and temporal analysis, on how deep SNNs successfully capture features within time-series data.

2. Related Work

2.1. Spiking Neural Networks

Different from traditional ANNs, SNNs utilize discrete spike trains instead of continuous floating-point values to transmit and compute information. According to Li et al. (2023), SNNs can be regarded as ANNs that incorporate bio-inspired spatiotemporal dynamics and utilize spiking activation functions (e.g., spiking neurons). Spiking neurons, such as Izhikevich neuron (Izhikevich, 2003) and Leaky Integrate-and-Fire (LIF) neuron (Maass, 1997), are usually applied to generate spike trains from floating-point values by a Heaviside step function.

Due to the non-differentiability of spike neurons, backpropagation (Rumelhart et al., 1986) can not be directly applied to train SNNs. Nowadays, there are two mainstream approaches to address this problem. Firstly, **ANN-to-SNN conversion** (Rueckauer et al., 2017; Hu et al., 2018) aims to convert weights of a well-trained ANN to its SNN counterpart by replacing the activation function with spiking neuron layers and adding scaling rules such as weight normalization and threshold constraints. Another popular approach is **direct training with surrogate gradients** (Wu et al., 2019), which introduces surrogate gradients during error backpropagation, enabling the entire procedure to be differentiable. Backpropagation through time (BPTT) (Werbos, 1990) is suitable for this approach, which applies the traditional backpropagation algorithm to the unrolled computational graph. In this paper, we choose direct training with surrogate gradients as our training method for its favorable attributes, namely the avoidance of an extensive number of time steps and the elimination of adjusting training objectives based on SNN architecture.

2.2. Time-Series Forecasting

Time-series forecasting plays a crucial role in data analysis, focusing on predicting future values based on historical observations. Early approaches, such as auto-regressive integrated moving average (ARIMA) (Box et al., 2015) and Gaussian Process (GP) (Roberts et al., 2013), primarily rely on statistical techniques. With the development of deep learning, methods based on convolutional neural networks (CNN) (Bai et al., 2018; Liu et al., 2022; Wu et al., 2023), recurrent neural networks (RNN) (Zhang et al., 2017; Siami-Namini et al., 2019), Transformer (Wu et al., 2021; Zhang & Yan, 2022; Liu et al., 2024), and graph neural networks (GNN) (Yu et al., 2018; Fang et al., 2023) have achieved great success on time-series forecasting task. Among these approaches, GNN-based methods predominantly focus on the spatial dimension rather than the temporal aspect, diverging from the emphasis of our proposed framework for SNNs. Consequently, our method exclusively involves the adaptation of CNNs, RNNs, and Transformers to their cor-

responding SNN counterparts.

Some studies have tried to apply SNNs in forecasting data of certain domains, such as financial data (Reid et al., 2014), wind power data (González Sopeña et al., 2022), and electricity data (Kulkarni et al., 2013). However, they either focus on how to deploy SNNs on neuromorphic hardware in real-world scenarios or fail to obtain satisfying performance due to simple architectures. Besides, there are works addressing time-series forecasting using spike neuron P system (Liu et al., 2021; Long et al., 2022), which is not an SNN but a distributed and parallel computing paradigm.

3. Methodology

3.1. Preliminaries

3.1.1. Task Formulation

We consider the regular time-series forecasting task where all the time series are sequences sampled from underlying continuous signals $\mathcal{X}(t)$ with constant discretization step size ΔT as $\mathbf{x}_k = \mathcal{X}(k\Delta T)$. Given the historical observed time series $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\} \in \mathbb{R}^{T\times C}$ for T time steps, the multivariate time-series forecasting task aims to predict the values in the subsequent L time steps $\mathbf{Y} = \{\mathbf{x}_{T+1}, \mathbf{x}_{T+2}, \ldots, \mathbf{x}_{T+L}\} \in \mathbb{R}^{L\times C}$, where C denotes the number of variates.

3.1.2. Spiking Neurons and Surrogate Gradients

The basic unit in SNNs is the leaky integrate-and-fire (LIF) neuron (Maass, 1997) which operates on an input current $I(t)$ and contributes to the membrane potential $U(t)$ and the spike $S(t)$ at time t. The dynamics of the LIF neuron shown in Figure 1 can be written as:

$$U(t) = H(t-\Delta t) + I(t), \quad I(t) = f(\mathbf{x};\theta), \tag{1}$$

$$H(t) = V_{reset}S(t) + (1 - S(t))\,\beta U(t), \tag{2}$$

$$S(t) = \begin{cases} 1, & \text{if } U(t) \geq U_{\text{thr}} \\ 0, & \text{if } U(t) < U_{\text{thr}} \end{cases}, \tag{3}$$

where $I(t)$ is the spatial input to the LIF neuron at time step t calculated by applying function f with $\mathbf{x}$ as input and θ as learnable parameters. $H(t)$ is the temporal output of the neuron at time step t and Δt is the discretization constant controlling the granularity of LIF modeling. The spike $S(t)$ is defined as a Heaviside step function depending on the membrane potential. When $U(t)$ achieves the threshold U_{thr}, the neuron will fire and emit a spike, then the temporal output $H(t)$ will be reset to V_{reset}. Otherwise, no spike will be emitted and the membrane potential $U(t)$ will decay to $H(t)$ by a decay rate β.

Now we generate the spike trains $\mathbf{S} \in \mathbb{R}^{T'\times N}$ with a spiking

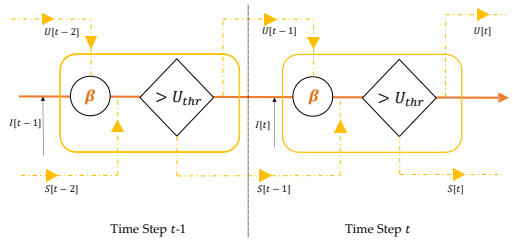


Figure 1. A recurrent representation of a leaky integrate-and-fire (LIF) neuron. The membrane potential $U(t-1)$ and spike $S(t-1)$ at time step $t-1$ are derived from their counterparts at time step $t-2$ and undergo processing to yield $U(t)$ and $S(t)$ at time step t.

neuron layer $\mathcal{SN}(\cdot)$:

$$\mathbf{S} = \mathcal{SN}(\mathbf{I}) \tag{4}$$

by iterating T' steps over N input currents $\mathbf{I} \in \mathbb{R}^{T'\times N}$ with N LIF neurons.

As mentioned in Section 2.1, we choose direct training with surrogate gradients as our method to train SNNs. we follow Fang et al. (2020b) to choose the arctangent-like surrogate gradients as our error estimation function when backpropagation, which regards the Heaviside step function (Equation 3) as:

$$S(t) \approx \frac{1}{\pi}\arctan(\frac{\pi}{2}\alpha U(t)) + \frac{1}{2} \tag{5}$$

where α is a hyper-parameter to control the frequency of the arctangent function. Therefore, the gradients of S in Equation 5 are $\frac{\partial S(t)}{\partial U(t)} = \frac{\alpha}{2}\frac{1}{(1+(\frac{\pi}{2}\alpha U(t))^2)}$ and thus the overall model can be trained in an end-to-end manner with back-propagation through time (BPTT).

3.2. Temporal Alignment and Spike Encoder

To utilize the intrinsic nature of SNN to its best, it's crucial to align the temporal dimension between time-series data and SNNs. Our central concept is to incorporate relevant finer information of the spikes within the time-series data at each time step. Specifically, we divide a time step ΔT of the time series into T_s segments and each of them allows a firing event for neurons whose membrane potentials surpass the threshold, i.e., $\Delta T = T_s\Delta t$.

This equation bridges between a time-series time step ΔT and an SNN time step Δt. As a result, the independent variable t in time-series ($\mathcal{X}(t)$) and in SNN ($U(t), I(t), H(t), S(t)$) are now sharing the same meaning. To this end, the spiking encoder, responsible for generating the first spike trains based on the floating-point inputs, needs to calculate $T_s \times T \times C$ possible spike events. The most straightforward non-parametric approach is to consider each

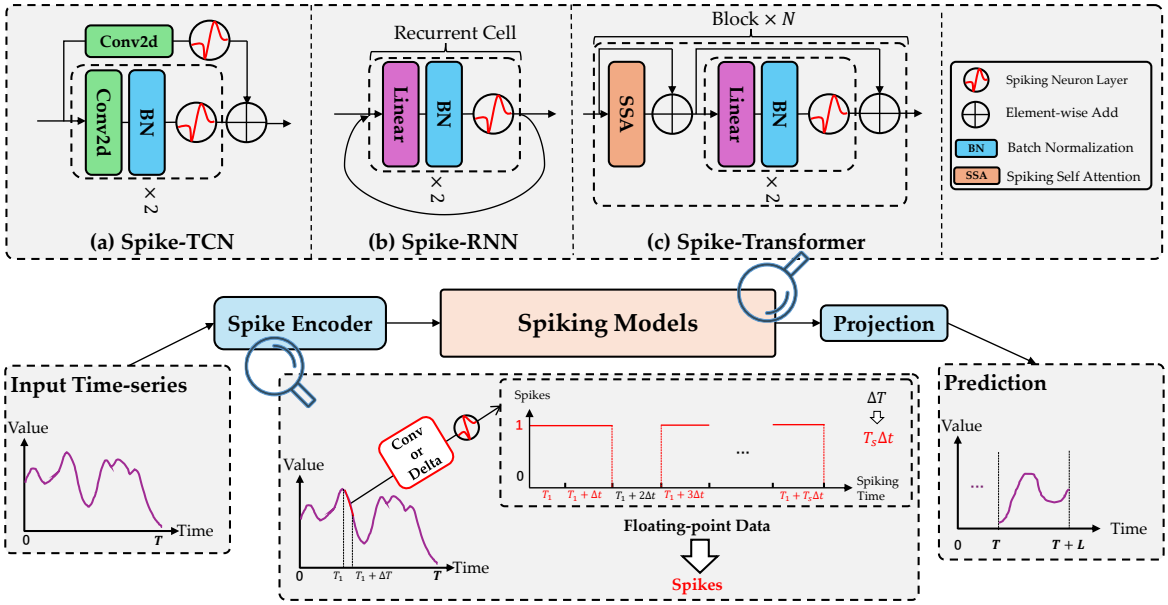


Figure 2. An overview of our framework for SNNs in time-series forecasting. Given an input time-series sample $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ with T, our goal is to predict the values in the following L time steps $\mathbf{Y} = \{\mathbf{x}_{T+1}, \mathbf{x}_{T+2}, \ldots, \mathbf{x}_{T+L}\}$. Firstly, a spike encoder will be used to generate spike trains with T_s spiking time steps from the original data every Δt time step. After being encoded, time-series data will be converted to spike trains ($B \times T_s \times T \times C$) and will be fed into SNNs. We provide three SNNs: (a) Spike-TCN; (b) Spike-RNN; and (c) Spike-Transformer. Finally, the spike trains will be converted to floating-point values by a projection layer.

data point in the input time series as the current value and replicate it T_s times. However, this approach can disrupt the continuous nature of the underlying $\mathcal{X}(t)$ hypothesis. Therefore, we seek to use parametric spike encoding techniques.

Delta Spike Encoder The delta spike encoder, originated from the delta modulation (Eshraghian et al., 2021), is inspired by the biological notion that neurons are sensitive to temporal changes. The mathematical expression governing this process is encapsulated as follows:

$$\mathbf{S} = \mathcal{SN}\left(\mathrm{BN}\left(\mathrm{Linear}\left(\mathbf{x}_t - \mathbf{x}_{t-1}\right)\right)\right) \tag{6}$$

where a linear layer is applied to the temporal differences to learn different sensitivities on different SNN time steps and expand the dimension of the spike train $\mathbf{S}$ to $T_s \times T \times C$. The result undergoes batch normalization (BN) and passed through a spiking neuron layer $\mathcal{SN}$ to be converted to spike trains.

Convolutional Spike Encoder In time-series tasks, the shapes of the sequence are often categorized as interpretable features (Ye & Keogh, 2009) for time-series classification and clustering. Recently, Qu et al. (2024) demonstrated that this kind of morphological information could be modeled by a particular type of CNN kernel. Therefore, we propose to use a convolutional layer as a suitable temporal encoder which should emit spikes as long as the shape of the original subsequences matches the kernel.

Given the historical observed time-series $\mathbf{X} \in \mathbb{R}^{T \times C}$, we feed it into a convolutional layer followed by batch normalization and generate the spikes as:

$$\mathbf{S} = \mathcal{SN}\left(\mathrm{BN}\left(\mathrm{Conv}\left(\mathbf{X}\right)\right)\right). \tag{7}$$

Similar to the delta spike encoder, by passing through the convolutional layer, the dimension of the spike train $\mathbf{S}$ is expanded to $T_s \times T \times C$. Spikes at every SNN time step are generated by pairing the data with different convolutional kernels.

Both the delta spike encoder and the convolutional spike encoder capture internal temporal information of the input data, i.e., temporal changes and shapes, respectively, contributing to the representation of the dynamic nature of the information over time and catering to the following spiking layers for event-driven modeling.

3.3. Spiking Model Architrcture

In this section, we discuss the temporal spiking neural network to model the obtained spike trains. We convert three distinct types of classic yet powerful temporal-oriented ANNs designed for time-series forecasting tasks, i.e., TCN, RNN (and the GRU variant), and iTransformer, to their respective SNN counterparts.

Spike-TCN Temporal Convolutional Network (TCN) (Bai et al., 2018) uses convolutional kernels to model time series. Unlike general CNNs, TCN can map any length

of the time series to the same length without information leakage from the future to the past. Inspired by practice from image classification (He et al., 2016), recent TCN also involves the residual connection to overcome the unstable gradient problems.

Following Hu et al. (2018), we construct the Spike-TCN by making the following changes to the original TCN: 1) We replace the ReLU activation function with a spiking neuron layer. This substitution is a characteristic feature of SNNs, where the firing of neurons is modeled in a more biologically plausible way. 2) We remove the dropout operation which is hardware-unfriendly. The dropout operation involves two steps: randomly zeroing some elements of the input tensor with a probability of p, and scaling the outputs by a factor of $\frac{1}{1-p}$. The second step introduces division operations, which are not hardware-friendly. 3) We replace the residual shortcut in vanilla TCN with the spike-element-wise (SEW) residual module (Fang et al., 2021), which implements identity mapping and overcomes the vanishing/exploding gradient problems in a spiking version. 4) The down-sampling module is also converted to its spiking version, which follows SEW rules.

Since TCN only involves local convolution and doesn't track temporal state across time steps, the membrane potentials in Spike-TCN $U(t)$ are set to 0 at the beginning of every time-series time step. This makes it possible for parallel training.

Spike-RNN The vanilla recurrent neural network (RNN) uses its internal state to process the sequence of inputs and can output a sequence of the same length iteratively. We rewrite the recurrent cell of the original RNN to construct the Spike-RNN by substituting the activation function with the spiking neuron layer. Unlike TCN, RNN tracks the temporal states and thus the membrane potential in Spike-RNN will persist across time steps. We also modify the gated recurrent unit (GRU) (Cho et al., 2014), a popular variant of RNN, which uses a gating mechanism to address the long-term dependency problem.

Spike-Transformer The use of the Transformer architecture in the time-series forecasting task attracts a massive amount of attention (Wu et al., 2021; Liu et al., 2024), yet no consensus has been reached on what the best framework to apply the self-attention operation. In this work, we build our spiking version of Transformer based on iTransformer (Liu et al., 2024) and name it "iSpikformer" considering two justifications: 1) iTransformer is the state-of-the-art time-series forecasting model on several public benchmarks and thus is strong enough to serve as our basis; 2) iTransformer treats the independent time series as tokens to capture multivariate correlations through the self-attention mechanism, which mainly focuses on spatial modeling across channels. By constructing its spiking counterpart, we demonstrate that our design of spikes to model temporal dynamics is essentially orthogonal to spatial modeling and can be further boosted with relevant advancements.

Currently, there are various spiking Transformers designed for image classification tasks (Li et al., 2022; Zhou et al., 2023b;a; Yao et al., 2023a). Among them, Spikformer v2 (Zhou et al., 2024) based on Spikformer achieved current state-of-the-art performance on Imagenet-1k and CIFAR-10 benchmarks. Therefore, we follow Spikformer to implement the spiking self-attention (SSA) mechanism and use it to replace the original self-attention layer in iTransformer to construct our spiking Transformer blocks.

Specifically, after the spike trains are obtained by the spike encoder detailed in Section 3.2, a channel-wise spiking embedding layer will be applied as:

$$\mathbf{S}_{emb} = \mathcal{SN}(\mathrm{Linear}(\mathbf{S})), \tag{8}$$

where $\mathbf{S}_{emb} \in \mathbb{R}^{H \times C}$ are C channel-wise embeddings of dimension H. These embeddings are afterward fed into the spiking Transformer blocks.

Note that the SNNs we have designed for time-series forecasting strictly adhere to hardware-friendly requirements. Specifically, the inference process of the model avoids involving floating-point operations, such as multiply-and-accumulate (MAC) operations. This design choice enables these models to be effectively deployed on neuromorphic chips, aligning with the hardware constraints and characteristics of such platforms.

3.4. Spike Decoding

After passing through the final spiking neuron layer, we obtain spiking hidden states represented as $\mathbf{S}_{hidden}$. In image classification tasks, a linear layer is commonly employed as a classification head to produce predictions. Similarly, in the context of time-series forecasting, which is essentially a regression task, we transform the spiking data into forecasting sequences by applying a fully connected layer, denoted as $\mathbf{Y} = \mathrm{Linear}(\mathbf{S}_{hidden})$. Since there are no additional floating-point operations applied to $\mathbf{Y}$ beyond this step, it can be accommodated within the framework of our design.

4. Experiments

In this section, we conduct experiments to investigate the following research questions:

RQ1: Encompassing the merits of energy efficiency, biological plausibility, and the event-driven paradigm, can these SNNs achieve comparable performance to their ANN counterparts?

RQ2: Is our design of temporal alignment and corresponding spike encoders effective and robust?

Table 1. Experimental results of time-series forecasting on 4 benchmarks with different prediction lengths (horizons) L. The best and the second-placed results are formatted in bold font and underlined format. ↑ (↓) indicates the higher (lower) the better. All SNNs are equipped with a convolutional spike encoder in this table. The numbers in the **Avg. Rank** column indicate the average ranking of the current row's models within each specific setting. Numbers in the **Avg.** column with * indicate that a model significantly ($p < 0.05$) outperforms its counterpart. All results are averaged across 3 random seeds.

Method	Spike	Metric	Metr-la				Pems-bay				Solar				Electricity				Avg.	Avg. Rank↓
			6	24	48	96	6	24	48	96	6	24	48	96	6	24	48	96		
ARIMA	✗	R^2↑	.687	.441	.282	.265	.741	.723	.692	.670	.951	.847	.725	.682	.963	.960	.914	.863	.713	7.3
		RSE↓	.575	.742	.889	.902	.532	.548	.562	.612	.202	.365	.588	.589	.522	.534	.564	.599	.583	7.3
GP	✗	R^2↑	.685	.437	.265	.233	.732	.712	.689	.665	.944	.836	.711	.675	.962	.968	.912	.852	.705	8.4
		RSE↓	.572	.738	.912	.925	.544	.532	.577	.592	.225	.388	.612	.575	.603	.612	.633	.642	.605	7.6
TCN	✗	R^2↑	.820	.601	.455	**.330**	.881	**.749**	**.695**	**.689**	.958	.871	.737	.661	.975	.973	.968	.962	.770	3.9
		RSE↓	.446	.665	.778	**.851**	.373	.541	.583	.587	.210	.359	.513	.583	.282	.287	**.319**	.345	.483	3.6
Spike-TCN	✓	R^2↑	.783	.603	**.468**	.326	.811	.729	.662	.633	.937	.840	.708	.650	.970	.963	.958	.953	.750	7.0
		RSE↓	.491	.665	.769	.865	.469	.541	.625	.635	.259	.401	.541	.596	.333	.342	.368	.389	.518	6.1
GRU	✗	R^2↑	.759	.429	.301	.194	.747	.703	.691	.665	.950	.875	.781	.737	**.981**	.972	.971	**.964**	.733	5.8
		RSE↓	.517	.797	.882	.947	.529	.573	.584	.608	.219	.355	.476	.522	.506	.598	.537	.587	.573	7.1
Spike-GRU	✓	R^2↑	**.846**	.615	.427	.275	.864	.741	.688	.657	.912	.822	.771	.668	.978	.964	.962	.959	.759	6.2
		RSE↓	.414	.663	.827	.943	.398	.535	.601	.621	.299	.430	.485	.629	.280	.317	.338	.484	.517*	6.0
Spike-RNN	✓	R^2↑	**.846**	.622	.433	.283	.872	.745	.685	.654	.923	.820	**.812**	.714	.977	.972	.962	.960	.768*	5.2
		RSE↓	**.412**	**.648**	.794	.935	.387	**.528**	.588	.634	.278	.425	**.435**	.586	.267	.296	.346	.481	.503*	4.8
Autoformer	✗	R^2↑	.762	.548	.411	.282	.782	.711	.689	.668	.960	.852	.791	.701	.980	**.977**	**.975**	.963	.753	4.6
		RSE↓	.565	.692	.785	.872	.452	.543	.577	**.565**	.212	.432	.622	.685	.481	.506	.566	.548	.569	6.6
iTransformer	✗	R^2↑	.829	**.623**	.439	.285	**.887**	.719	.685	.668	**.964**	**.879**	.799	**.738**	.979	**.977**	**.975**	**.964**	**.776**	**2.9**
		RSE↓	.436	**.648**	.780	.878	**.362**	.547	**.561**	.584	**.191**	.348	.448	.563	**.259**	.305	.335	.427	.480	**2.8**
iSpikformer	✓	R^2↑	.817	.618	.440	.279	.879	.744	.687	.674	.961	.876	.795	**.738**	.977	.974	.972	.963	.775	3.5
		RSE↓	.475	.668	**.752**	.905	.376	.536	.569	.580	.204	**.333**	.465	**.521**	.263	**.284**	.338	**.348**	**.476**	2.9

RQ3: To what extent do the SNNs help reduce energy consumption?

4.1. Experiment Settings

To assess the forecasting capabilities of the compared methods, the datasets listed below are employed: **Metr-la** (Li et al., 2017b): Average traffic speed measured on the highways of Los Angeles County; **Pems-bay** (Li et al., 2017b): Average traffic speed in the Bay Area; **Electricity** (Lai et al., 2018): Hourly electricity consumption measured in kWh; **Solar** (Lai et al., 2018): Records of solar power production.

On these forecasting datasets, we compare our method with two statistics methods **ARIMA** (Box et al., 2015) and **GP** (Roberts et al., 2013); one CNN-based model **TCN** (Bai et al., 2018); one RNN-based model **GRU** (Cho et al., 2014); two Transformer-based models **Autoformer** (Wu et al., 2021) and **iTransformer** (Liu et al., 2024).

For evaluating time-series forecasting tasks, we employ the Root Relative Squared Error (RSE) and the coefficient of determination (R^2). The detailed statistics of datasets, hyperparameters of models, and formulation of the evaluation metrics can be referred to Appendix A.

4.2. Main Results

We report the results of all the methods on 4 time-series forecasting tasks with various prediction lengths L in Table 1. Results for ARIMA, GP, GRU, and Autoformer are obtained from the study conducted by Fang et al. (2023).

To conclude from Table 1, all the SNNs successfully model

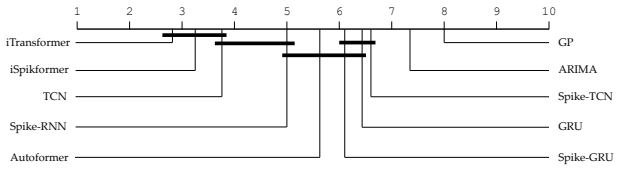


Figure 3. Critical Difference (CD) diagram of all methods in Table 1 on time series forecasting tasks with a confidence level of 95%.

the time series and achieved reasonable performances, and our iSpikformer achieves comparable or even better performance compared to the state-of-the-art ANN model, answering **RQ1**. To elaborate:

(1) SNNs succeed when temporal dynamics are properly preserved and handled. While Spike-TCN underperforms TCN, Spike-GRU, and Spike-RNN achieved significantly better performances over the GRU baseline. We attribute this phenomenon to the different ability of the models to handle sequential dynamics. Similar to TCN, Spike-TCN models only local features of the sequence and doesn't track the temporal states. As a result, the membrane potentials are hardly reset to 0 across time steps, which violates the nature of SNNs. On the other hand, Spike-RNN and Spike-GRU persist in such temporal information and cater to our overall event-driven paradigm. It is worth highlighting that Spike-RNN can achieve state-of-the-art performance on benchmarks with both long and short historical observations. Furthermore, by examining the critical difference diagram depicted in Figure 3, we can roughly categorize all the methods we have employed into three tiers, and it was a pleasant surprise to observe that our iSpikformer ranks

within the top tier.

(2) SNNs with our temporal modeling can be further improved by advanced spatial modeling techniques. One of the standout findings in our study is the exceptional performance of iSpikformer, which exhibits results nearly indistinguishable from the state-of-the-art ANN model, iTransformer. As observed in Table 1, iSpikformer achieves the lowest average RSE compared to all other methods and it nearly matches the R^2 performance of iTransformer, with only a marginal decrease of 0.001. Furthermore, it's noteworthy that the RSE of iSpikformer consistently ranks as either the lowest or the second lowest across all benchmark datasets. These findings confirm that our temporal modeling techniques perfectly complement the most advanced spatial modeling technique, and together the spiking neural network can be very effective to address the general time-series forecasting task.

Table 2. Performance of Spike-TCN, Spike-RNN, and iSpikformer on two benchmarks with different encoder types. Numbers with $*$ indicate that SNNs fail to converge in the settings. Numbers presented in **bold** font indicate that this particular type of encoder has achieved the best performance within the SNN model.

Method	Encoder	Metric	Metr-la				Electricity			
			6	24	48	96	6	24	48	96
Spike-TCN	Convolutional	$R^2\uparrow$	**.783**	**.603**	**.468**	**.326**	**.970**	**.963**	**.958**	**.953**
		RSE$\downarrow$	**.491**	**.664**	.769	.935	**.333**	**.342**	**.368**	**.389**
	Delta	$R^2\uparrow$	.751	.582	.458	.317	.963	.956	.948	.942
		RSE$\downarrow$	.525	.676	**.768**	**.871**	.344	.371	.432	.460
	Repetition	$R^2\uparrow$	.024$*$	.024$*$	.022$*$	.020$*$	.878	.710$*$	.710$*$	.710$*$
		RSE$\downarrow$	1.05$*$	1.04$*$	1.04$*$	1.05$*$	.662	1.03$*$	1.03$*$	1.03$*$
Spike-RNN	Convolutional	$R^2\uparrow$	**.846**	**.622**	**.433**	**.283**	**.977**	**.972**	**.962**	**.960**
		RSE$\downarrow$	**.412**	**.648**	**.794**	**.935**	**.267**	**.296**	.346	**.481**
	Delta	$R^2\uparrow$	.839	.616	.430	.277	.969	.966	.962	.876
		RSE$\downarrow$	.420	.652	.799	.938	.301	.318	**.344**	.685
	Repetition	$R^2\uparrow$	.817	.578	.021$*$	.021$*$	.901	.816	.710$*$	.710$*$
		RSE$\downarrow$	.481	.684	1.04$*$	1.04$*$	.592	.766	1.03$*$	1.04$*$
iSpikformer	Convolutional	$R^2\uparrow$	**.817**	**.618**	**.440**	**.279**	**.977**	**.974**	**.972**	**.963**
		RSE$\downarrow$	**.475**	.668	**.752**	**.905**	**.263**	**.284**	**.338**	**.348**
	Delta	$R^2\uparrow$	.804	.601	.434	.272	.972	.969	.960	.944
		RSE$\downarrow$	.496	**.666**	.759	.910	.274	.302	.391	.455
	Repetition	$R^2\uparrow$	.692	.548	.238	.021$*$	.962	.953	.849	.710$*$
		RSE$\downarrow$	.573	.708	.847	1.04$*$	.289	.557	.705	1.03$*$

4.3. Model Analysis

We conduct model analysis experiments to answer **RQ2**, i.e., to verify whether our proposed methods are effective and robust.

4.3.1. Encoder Type

As discussed in Section 3.2, spike encoders are used to convert continuous time-series data to spike trains. To verify the effectiveness of our proposed spike encoder, we evaluate the performance of Spike-TCN, Spike-RNN, and iSpikformer with three different types of encoders: CNN-based encoder, Delta-based encoder, and Repetition encoder. Repetition encoder refers to the most widely-used encoding methods in previous SNN studies, where the time-series data is repeated T_s times to add an extra dimension. In this experiment, we focus on Metr-la and Electricity datasets, taking into account the different lengths of historical observations. The results of this comparison are presented in Table 2.

In summary, our proposed spike encoders, including the convolutional spike encoder, and delta spike encoder, have demonstrated their effectiveness in capturing temporal information from time-series data. Specifically: (1) While repetition is a common strategy utilized in many tasks, it appears that SNNs with repetition encoders may struggle to converge under many settings in time-series forecasting tasks. This confirms the necessity of an exquisite temporal modeling technique. (2) Both convolutional spike encoder and delta spike encoder are event-driven spike generators, however, the performance of SNNs utilizing convolutional spike encoders surpasses that of SNNs employing delta spike encoders by an increase of 0.09 on average. This improvement demonstrates that the shape-based encoder which takes a wide scope of sequence into consideration is more effective than a change-based encoder which reacts to only the very local changes.

4.3.2. Hyper-parameters

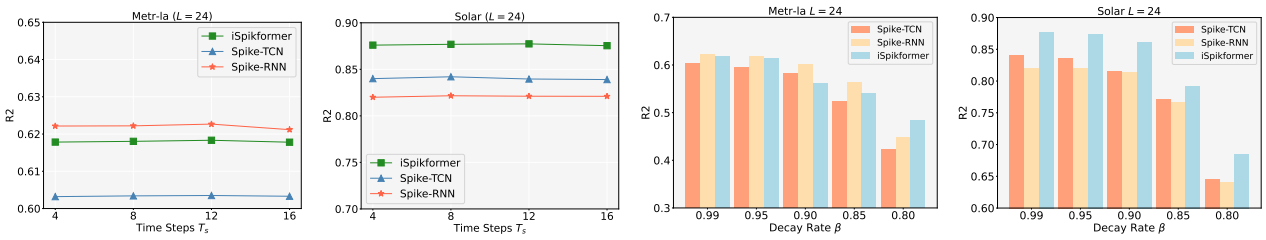
To verify the robustness of our design, we investigate how sensitive the performances respond to different choices of time-related hyper-parameters.

Time Step. As introduced in Section 3.2, time step T_s in SNNs is a hyper-parameter that controls how accurately the SNNs model the temporal dynamics Δt. We conduct experiments with varying values of T_s from the set $\{4, 8, 12, 16\}$ on the Metr-la and Solar benchmarks, both with a horizon of $L = 24$. Based on the results presented in Figure 4 (a) and (b), we observe that, in general, the R^2 values remain relatively stable with minimal variation as T_s increases. However, there may be a slight decrease in R^2 when $T_s = 16$. This phenomenon can be reasonably explained by the occurrence of "*self-accumulating dynamics*" (Fang et al., 2020c). This refers to an error accumulation triggered by surrogate gradients, which can lead to gradient vanishing or explosion, potentially affecting the model's performance adversely.

Decay Rate. We performed experiments using different values of β taken from the set $\{0.99, 0.95, 0.90, 0.85, 0.80\}$ on the Metr-la and Solar benchmarks, both with a forecasting horizon of $L = 24$. As illustrated in Figure 4 (c) and (d), it is evident that when the value of β increases, there is a noticeable decrease in R^2 performance on both the Metr-la and Solar benchmarks. This observation can be attributed to the fact that a higher β makes the SNN more persistent in its internal state, which is beneficial for retaining long-term information.

4.4. Energy Reduction

An essential advantage of SNNs is the low consumption of energy during inference. Assuming that we run Spike-TCN,



(a) (b) (c) (d)

Figure 4. The impact of two crucial hyper-parameters in SNNs: time Steps T_s and the decay rate β. (a) and (b): R^2 versus T_s on Metr-la and Solar respectively. (c) and (d): R^2 versus β on Metr-la and Solar respectively. The horizon L of these experiments is set to 24.

Spike-RNN, and iSpikformer on a 45nm neuromorphic hardware (Horowitz, 2014), we can calculate the theoretical energy consumption (Appendix B). We compare the theoretical energy consumption per sample of our proposed three SNNs and their original ANN counterparts on test sets of Electricity benchmarks with $L = 24$ during inference (Table 3).

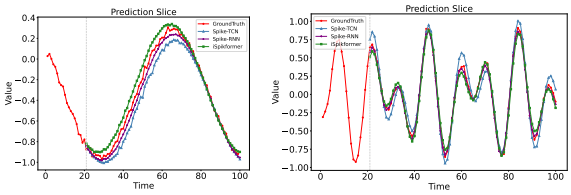
Table 3. Theoretic energy consumption per sample of Electricity during the inference stage. "OPs" refers to SOPs in SNN and FLOPs in ANN. "SOPs" denotes the synaptic operations of SNNs. "FLOPs" denotes the floating point operations of ANNs.

Model	Param(M)	OPs (G)	Energy (mJ)	Energy Reduction	R^2
TCN	0.460	0.14	0.64	**63.60%** ↓	.973
Spike-TCN	0.461	0.15	0.23		.963
GRU	1.288	1.32	6.07	**75.05%** ↓	.972
Spike-GRU	1.289	1.63	1.51		.964
iTransformer	1.634	2.05	9.47	**66.30%** ↓	.977
iSpikformer	1.634	3.55	3.19		.974

As shown in Table 3, SNNs exhibit a notably lower energy consumption compared to their ANN counterparts, resulting in an average decrease of approximately 70.33%, which answers **RQ3**. This compelling advantage positions SNNs as an attractive choice for energy-efficient solutions. It is worth emphasizing that the parameters of an SNN and an ANN are nearly identical, as we only eliminate certain parameter-free operations, such as layer normalization and dropout when converting an ANN to its SNN version. Notably, Spike-GRU demonstrates a significant reduction in energy consumption, reaching up to 75.05% less compared to the traditional GRU model. This observation suggests that Spike-GRU exhibits the lowest average firing rate of spiking neurons and the most sparse data flow among the considered models, contributing to its energy efficiency.

4.5. Temporal Analysis

To validate the capability of the proposed SNNs for time series prediction in capturing the time variation characteristics of time series data, we conducted experiments on both low-frequency and high-frequency *synthetic periodic signals* using Spike-TCN, Spike-RNN, and iSpikformer, as illustrated in Figure 5. Details of the synthetic periodic signals can be found in Appendix A.2



(a) (b)

Figure 5. A prediction slice ($T = 20, L = 80$) of Spike-TCN, Spike-RNN, and iSpikformer on synthetic time-series data. (a) Prediction slice on low-frequency data. (a) Prediction slice on high-frequency data.

Based on the observations from Figure 5, we can draw the following conclusions: (1) Spike-TCN, Spike-RNN, and iSpikformer all demonstrate strong performance when dealing with both low-frequency and high-frequency time-series data; (2) When dealing with high-frequency signals, Spike-TCN seems to exhibit lower accuracy in predicting peak values compared to Spike-RNN and iSpikformer. This observation could potentially account for the relatively weaker overall performance of Spike-TCN within our proposed SNNs. Refer to Appendix C for visualizations of forecasting results on real datasets.

5. Conclusion

In this paper, we present a framework for utilizing SNNs in time-series forecasting tasks. Through a series of experiments, we have demonstrated the efficacy of our proposed SNN-based approaches in time-series forecasting. These approaches have shown comparable performance to traditional time-series forecasting methods across diverse benchmark datasets, while significantly reducing energy consumption. Furthermore, our detailed analysis experiments have shed light on the SNN's ability to capture temporal dependencies within time-series data. This insight underscores the nuanced strengths and effectiveness of SNNs in modeling

the intricate dynamics of time series. In summary, our study contributes to the expanding field of SNNs, offering an energy-efficient and biologically plausible alternative for time-series forecasting tasks. The limitations and future directions are discussed in Appendix D.

Impact Statement

Our research contributes to the growing field of spiking neural networks and presents a promising alternative for time-series forecasting tasks. It paves the way for the development of more biologically inspired and temporally aware forecasting models, offering exciting prospects for future advancements in this domain. Besides, we do not think our work will have a bad impact on ethical aspects and future societal consequences.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments. This work was supported partially by National Natural Science Foundation of China (No. 62076068).

References

Amir, A., Taba, B., Berg, D. J., Melano, T., McKinstry, J. L., di Nolfo, C., Nayak, T. K., Andreopoulos, A., Garreau, G. J., Mendoza, M., Kusnitz, J. A., DeBole, M. V., Esser, S. K., Delbrück, T., Flickner, M., and Modha, D. S. A low power, fully event-based gesture recognition system. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7388–7397, 2017.

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

Chen, X., Wu, J., Tang, H., Ren, Q., and Tan, K. C. Unleashing the potential of spiking neural networks for sequential modeling with contextual embedding. *arXiv preprint arXiv:2308.15150*, 2023.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, pp. 1724–1734, 2014.

Ding, J., Yu, Z., Tian, Y., and Huang, T. Optimal ANN-SNN conversion for fast and accurate inference in deep spiking neural networks. In Zhou, Z. (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*, pp. 2328–2336, 2021.

Dominguez-Morales, J. P., Liu, Q., James, R., Gutierrez-Galan, D., Jimenez-Fernandez, A., Davidson, S., and Furber, S. Deep spiking neural network model for time-variant signals classification: a real-time speech recognition approach. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2018.

Eshraghian, J. K., Ward, M., Neftci, E. O., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, Jeong, D. S., and Lu, W. D. Training spiking neural networks using lessons from deep learning. *ArXiv*, abs/2109.12894, 2021.

Fang, H., Shrestha, A., and Qiu, Q. Multivariate time series classification using spiking neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. IEEE, 2020a.

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., Tian, Y., and other contributors. Spikingjelly, 2020b.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2641–2651, 2020c.

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. Deep residual learning in spiking neural networks. In *Neural Information Processing Systems*, 2021.

Fang, Y., Ren, K., Shan, C., Shen, Y., Li, Y., Zhang, W., Yu, Y., and Li, D. Learning decomposed spatial relations for multi-variate time-series modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 7530–7538, 2023.

González Sopeña, J. M., Pakrashi, V., and Ghosh, B. A spiking neural network based wind power forecasting model for neuromorphic devices. *Energies*, 15(19):7256, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Horowitz, M. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Conference on Solid-State Circuits Conference, ISSCC 2014, Digest of Technical Papers*, pp. 10–14. IEEE, 2014.

Hu, Y., Tang, H., and Pan, G. Spiking deep residual networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

Izhikevich, E. M. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14 6:1569–72, 2003.

Jeffares, A., Guo, Q., Stenetorp, P., and Moraitis, T. Spike-inspired rank coding for fast and accurate recurrent neural networks. In *International Conference on Learning Representations*, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kulkarni, S., Simon, S. P., and Sundareswaran, K. A spiking neural network (snn) forecast engine for short-term electrical load forecasting. *Applied Soft Computing*, 13 (8):3628–3635, 2013.

Lai, G., Chang, W.-C., Yang, Y., and Liu, H. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 95–104, 2018.

Leñero-Bardallo, J. A., Serrano-Gotarredona, T., and Linares-Barranco, B. A 3.6 μs latency asynchronous frame-free event-driven dynamic-vision-sensor. *IEEE Journal of Solid-State Circuits*, 46(6):1443–1455, 2011.

Li, G., Deng, L., Tang, H., Pan, G., Tian, Y., Roy, K., and Maass, W. Brain inspired computing: A systematic survey and future trends. *Authorea Preprints*, 2023.

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. Cifar10-dvs: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11, 2017a.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017b.

Li, Y., Lei, Y., and Yang, X. Spikeformer: A novel architecture for training high-performance low-latency spiking neural network. *arXiv preprint arXiv:2211.10686*, 2022.

Liu, M., Zeng, A., Chen, M., Xu, Z., Lai, Q., Ma, L., and Xu, Q. Scinet: Time series modeling and forecasting with sample convolution and interaction. *Advances in Neural Information Processing Systems*, 35:5816–5828, 2022.

Liu, Q., Long, L., Peng, H., Wang, J., Yang, Q., Song, X., Riscos-Núñez, A., and Pérez-Jiménez, M. J. Gated spiking neural p systems for time series forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Liu, Y., Hu, T., Zhang, H., Wu, H., Wang, S., Ma, L., and Long, M. itransformer: Inverted transformers are effective for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024.

Long, L., Liu, Q., Peng, H., Wang, J., and Yang, Q. Multivariate time series forecasting method based on nonlinear spiking neural p systems and non-subsampled shearlet transform. *Neural Networks*, 152:300–310, 2022.

Lv, C., Li, T., Xu, J., Gu, C., Ling, Z., Zhang, C., Zheng, X., and Huang, X. Spikebert: A language spikformer learned from bert with knowledge distillation. 2023a.

Lv, C., Xu, J., and Zheng, X. Spiking convolutional neural networks for text classification. In *The Eleventh International Conference on Learning Representations*, 2023b.

Maass, W. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 14: 1659–1671, 1997.

Qu, E., Wang, Y., Luo, X., He, W., and Li, D. CNN kernels can be the best shapelets. In *The Twelfth International Conference on Learning Representations*, 2024.

Reid, D., Hussain, A. J., and Tawfik, H. Financial time series prediction using spiking neural networks. *PloS one*, 9(8):e103656, 2014.

Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N., and Aigrain, S. Gaussian processes for time-series modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371 (1984):20110550, 2013.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11, 2017.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Siami-Namini, S., Tavakoli, N., and Namin, A. S. The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International Conference on Big Data*, pp. 3285–3292. IEEE, 2019.

Werbos, P. J. Backpropagation through time: What it does and how to do it. *Proc. IEEE*, 78:1550–1560, 1990.

Wu, H., Xu, J., Wang, J., and Long, M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

Wu, H., Hu, T., Liu, Y., Zhou, H., Wang, J., and Long, M. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 1311–1318, 2019.

Yao, M., Hu, J., Zhou, Z., Yuan, L., Tian, Y., Bo, X., and Li, G. Spike-driven transformer. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a.

Yao, M., Zhao, G., Zhang, H., Hu, Y., Deng, L., Tian, Y., Xu, B., and Li, G. Attention spiking neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(8): 9393–9410, 2023b.

Ye, L. and Keogh, E. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 947–956, 2009.

Yu, B., Yin, H., and Zhu, Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pp. 3634–3640, 2018.

Zhang, X., Shen, F., Zhao, J., and Yang, G. Time series forecasting using gru neural network with multi-lag after decomposition. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part V 24*, pp. 523–532. Springer, 2017.

Zhang, Y. and Yan, J. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2022.

Zhou, C., Yu, L., Zhou, Z., Zhang, H., Ma, Z., Zhou, H., and Tian, Y. Spikingformer: Spike-driven residual learning for transformer-based spiking neural network. *arXiv preprint arXiv:2304.11954*, 2023a.

Zhou, Z., Zhu, Y., He, C., Wang, Y., Yan, S., Tian, Y., and Yuan, L. Spikformer: When spiking neural network meets transformer. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023b.

Zhou, Z., Che, K., Fang, W., Tian, K., Zhu, Y., Yan, S., Tian, Y., and Yuan, L. Spikformer v2: Join the high accuracy club on imagenet with an snn ticket. *arXiv preprint arXiv:2401.02020*, 2024.

A. Experiment Settings

A.1. Statistics of Datasets

We partitioned the forecasting datasets into train, validation, and test sets following a chronological order. The statistical characteristics and specific split details can be found in Table 4.

Dataset	Samples	Variables	Length	Train-Valid-Test Ratio
Metr-la	$34,272$	207	12	$(0.7, 0.2, 0.1)$
Pems-bay	$52,116$	325	12	$(0.7, 0.2, 0.1)$
Solar-energy	$52,560$	137	168	$(0.6, 0.2, 0.2)$
Electricity	$26,304$	321	168	$(0.6, 0.2, 0.2)$

Table 4. The statistics of datasets.

A.2. Synthetic Dataset

The synthetic dataset used in Section 4.5 is a uni-variate time series in the format of

$$\mathcal{X}(t) = A_1 \sin(\omega_1 t) + A_2 \sin(\omega_2 t + \phi) + \mathcal{N}(0, \sigma). \quad (9)$$

The synthetic time-series data is the combination of 3 independent terms. The first term uses a small $\omega_1 = 5 \times 10^{-3}$ to emulate the trend of the time series. We control the ω_2 in the second term to set the seasonality to different frequencies. The third term adds a Gaussian noise to the sequence as the residual. For low-frequency data, we set $A_1 \sim \mathrm{Uni}(1, 5)$, $A_2 \sim \mathrm{Uni}(1, 2)$, $\omega_2 = 0.04\pi$, and $\sigma = 0.3$. For high-frequency data, we set $A_1 = 9$, $A_2 = 8$, $\omega_2 = 0.1\pi$, $\phi \sim \mathrm{Uni}(0, 10)$, and $\sigma = 0.5$.

A.3. Implementation Details

To construct our proposed SNNs, we use two Pytorch-based frameworks: SnnTorch (Eshraghian et al., 2021) and SpikingJelly (Fang et al., 2020b). For all SNNs, we set the time step $T_s = 4$. For all LIF neurons in SNNs, we set threshold $U_{thr} = 1.0$, decay rate $\beta = 0.99$, $\alpha = 2$ in surrogate gradient function.

Spike-TCN For spike temporal blocks, we set the kernel size to 3 (short term) or 16 (long term), the output channel to 16, and the downsampling channel to 1. The time step of SNNs is set to 4. We totally use 3 blocks to construct our Spike-TCN.

Spike-RNN In the recurrent cells, we have configured two linear layers with input and output dimensions set to 128. The time step of SNNs is set to 4. This setup is consistent across the GRU cell as well.

iSpikformer We set the number of Spikformer blocks to 2, and the threshold of spiking neurons in the spiking self-attention (SSA) module as 0.25. The time step of SNNs is set to 4. Furthermore, we set the dimensional spike-form feature D as 512, and the hidden feature dimension of the feed-forward layer as 1024.

Training hyper-parameters we set the batch size as 128 and adopt Adam (Kingma & Ba, 2014) optimizer with a learning rate of 1×10^{-4}. We adopt an early stopping strategy with 30 epochs tolerance. We run our experiments on 4 NVIDIA RTX A6000 GPUs.

A.4. Evaluation metrics

In order to evaluate our model performances, we employ the Root Relative Squared Error (RSE) and the coefficient of determination (R^2) calculated as:

$$\mathrm{RSE} = \sqrt{\frac{\sum_{m=1}^{M} ||\mathbf{Y}^m - \hat{\mathbf{Y}}^m||^2}{\sum_{m=1}^{M} ||\mathbf{Y}^m - \bar{\mathbf{Y}}||^2}}, \quad (10)$$

$$R^2 = \frac{1}{MCL} \sum_{m=1}^{M} \sum_{c=1}^{C} \sum_{l=1}^{L} \left[1 - \frac{(Y_{c,l}^m - \hat{Y}_{c,l}^m)^2}{(Y_{c,l}^m - \bar{Y}_{c,l})^2} \right]. \quad (11)$$

In these equations, M represents the size of the test sets, C is the number of channels, and L is the prediction length. $\bar{\mathbf{Y}}$ is the average of $\mathbf{Y}^m$, $Y_{c,l}^m$ denotes the l-th future value of the c-th variable for the m-th sample, $\bar{Y}_{c,l}$ is the average of $Y_{c,l}^m$ across all samples, and $\hat{\mathbf{Y}}^m$ and $\hat{Y}_{c,l}^m$ denote the ground truths.

Compared to Mean Squared Error (MSE) or Mean Absolute Error (MAE), these metrics are more robust to the absolute values of the datasets and thus widely used in the time-series forecasting setting.

B. Theoretical Energy Consumption Calculation

According to Horowitz (2014) and Yao et al. (2023b), for SNNs, the theoretical energy consumption of layer l can be calculated as:

$$Energy(l) = E_{AC} \times SOPs(l) \quad (12)$$

where SOPs is the number of spike-based accumulate (AC) operations. For traditional artificial neural networks (ANNs), the theoretical energy consumption required by the layer b can be estimated by

$$Energy(b) = E_{MAC} \times FLOPs(b) \quad (13)$$

where FLOPs is the floating point operations of b, which is the number of multiply-and-accumulate (MAC) operations. We assume that the MAC and AC operations are implemented on the 45nm hardware (Yao et al., 2023b), where

$E_{MAC} = 4.6pJ$ and $E_{AC} = 0.9pJ$. Note that 1J $= 10^3$ mJ $= 10^{12}$ pJ. The number of synaptic operations at the layer l of an SNN is estimated as

$$SOPs(l) = T \times \gamma \times FLOPs(l) \tag{14}$$

where T is the number of times step required in the simulation, γ is the firing rate of the input spike train of layer l.

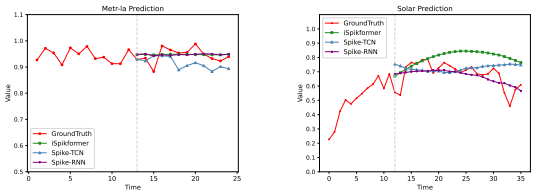
We want to emphasize that, although this approach to estimating power consumption has been used in many SNN algorithm papers, it is too simplified to show the real power consumption of SNNs in real-world scenarios. Yao et al. (2023b) think that although this estimation of energy consumption ignores the hardware implementation basis and the temporal dynamic of spiking neurons, it is still useful for simple analysis and evaluation of algorithm performance and guidance for algorithm design.

Besides, the inference speed of SNNs can not be estimated correctly **on GPUs** because the values 0 and 1 will still be processed to the **float-32** format. Once an SNN is well-trained, it can be deployed on neuromorphic hardware for inference, where 0 and 1 are treated as spike events. This enables significantly faster inference speed compared to traditional ANN implementations. If we just run SNNs on GPUs, then the inference speed of SNNs will always be T_s times that of ANNs, where T_s is the number of time steps of SNNs.

C. Case Study

In this section, we will show the ground truth and prediction of our SNNs in Metr-la ($L = 24$) and Solar ($L = 24$) Before we dive into the case study, it's crucial to emphasize that these two datasets present formidable challenges for prediction. This difficulty stems from the fact that each sample in these datasets includes a significant number of missing values, primarily occurring during nighttime periods when both traffic and solar energy tend to approach zero levels.

As depicted in 6 (a) and (b), all three models exhibit the capability to effectively capture the change trends within the non-missing value portion of the Metr-la dataset. Additionally, these models can roughly simulate the change trends in the Solar dataset as well. The ability of our proposed SNN model to accomplish this on such a complex dataset underscores its capacity to capture important time series features.



(a) (b)

Figure 6. Case Study (a) A prediction slice ($T = 12, L = 24$) of Spike-TCN, Spike-RNN, and iSpikformer on Metr-la. (b) A prediction slice ($T = 168, L = 24$) of Spike-TCN, Spike-RNN, and iSpikformer on Solar.

D. Limitations and Future Directions

D.1. Limitations.

Spike Decoder. The limitations of our work can be summarized as follows. In the case of SNNs for classification tasks, the use of a fully connected layer for predicting classification results is common. The classification results can be derived by counting the firing rate of spike trains within each classification category. Consequently, previous research often omitted the spiking neuron layers added to the projection layer. However, in regression tasks like time-series forecasting, we rely on the floating-point output of the final projection layer. In future work, we aim to explore ways to bridge this gap and potentially find methods that allow for a more unified approach in both classification and regression tasks within the SNN framework.

Architecture Designing. The study appears as an **engineering study**, i.e., replacing different parts of the model and evaluating how the model performance will be affected. In the long run, it would be much more insightful to consider a deeper **theoretical analysis** to tightly connect the encoder, spiking network model, and decoder together. Our desired framework is decoupled. The design of our spike encoders is inspired by the concept of temporal alignment (see Section 3.2). The architecture of our SNNs benefits from ANNs. Experimental results validate the effectiveness of each component, and good performance is achieved across various backbones (CNN, RNN, Transformers). Our motivation is to introduce a unified framework for SNNs tailored specifically for time-series forecasting tasks. What's more, current SNN works on model architecture can be roughly divided into two parts: (1) designing SNN modules inspired by biological theories and principles; (2) adapting existing ANN modules to suit the requirements of SNNs. We agree that while the former is theoretically robust, it poses significant challenges in terms of design and evaluation, whereas the latter offers a more accessible route for SNN development, hence its popularity among researchers. A fundamental aspect of our

contribution lies in bridging the gap between ANNs and SNNs, thereby providing a standardized guideline for the broader community. Moving forward, we intend to delve deeper into theoretical analyses to elucidate the intricate connections between various SNN modules, building upon the insights garnered from our empirical investigations.

D.2. Future directions.

Future research directions encompass: (1) Developing hardware-friendly algorithms that enable parallel training of TCN-like models without the need to reset $U(t)$ to 0. (2) Exploring more promising methods to enhance the utilization of SNNs for capturing **spatial information** within time-series data, such as spiking graph neural networks.