Decision SpikeFormer: Spike-Driven Transformer for Decision Making

Supplementary Material

A. Notation and Definitions

In this section, we provide a summary of the notation and definitions used throughout the paper and appendix, as outlined in Table 1.

B. Spiking Neuron Network

In this section, we present the surrogate gradient training method used in this work and describe how it is integrated into the backpropagation process for optimizing our loss function in DSFormer. We also provide an overview of additional Related Work in Offline RL and spike-driven Transformers.

B.1. Surrogate Gradient and Loss Backpropagation

Surrogate Gradient The non-differentiable dynamics of spiking neural networks make traditional backpropagation algorithms unsuitable for direct application. To address this, surrogate gradient methods have been developed to approximate the gradients of SNNs [20, 21, 33], enabling effective training. In our work, we adopt the surrogate gradient method proposed by [33], formulated as:

$$\frac{\partial S^t}{\partial U^t} \approx \frac{1}{\Delta} \operatorname{sign}\left(\left|U^t - U_{\text{th}}\right| \leq \frac{\Delta}{2}\right) \tag{1}$$

where Δ is a hyperparameter used to ensure the gradient integral is 1 and to control the slope of the function. We set $\Delta = 0.5$ in our experiments.

Loss Backpropagation in DSFormer We inherit the loss function in the Decision Transformer to our work:

$$\mathcal{L} = \frac{1}{L}\sum_{l=1}^{L}(\hat{a}_l - a_l)^2 \tag{2}$$

$$\hat{a}_l = W_{head}(\frac{1}{T}\sum_{t=1}^{T} S_{l,last}^t) + b_{head} \tag{3}$$

where $\hat{a}_l$ is the predicted action at iteration step l, a_l is the ground truth action at iteration step l, $S_{l,last}^t$ is the spike output of the last Spike Neuron at time step t, W_{head} and b_{head} are the weight matrix and bias of Prediction Head, respectively, and L is the total iteration steps. Following the surrogate gradient training method, the loss backpropagation of spiking neural networks can be formulated as:

$$\frac{\partial \mathcal{L}}{\partial W_{head}} = \sum_{l=1}^{L} \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{a}_l}\frac{\partial \hat{a}_l}{\partial W_{head}}}_{\text{constant}} \tag{4}$$

For other linear layers, the input X undergoes a linear transformation WX before being fed into spiking neurons, resulting in the corresponding membrane potential U and spikes S. According to [23], the gradient of the loss function with respect to the weight matrix W can be computed as:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_i \frac{\partial \mathcal{L}}{\partial S^i}\underbrace{\frac{\partial S^i}{\partial U^i}}_{\text{constant}}\sum_{j\leq i}\frac{\partial U^i}{\partial W^j} \tag{5}$$

$$\sum_{j\leq i}\frac{\partial U^i}{\partial W^j} = \underbrace{\frac{\partial U^i}{\partial W^i}}_{\text{constant}} + \underbrace{\frac{\partial U^i}{\partial U^{i-1}}}_{\text{constant}}\sum_{j\leq i-1}\frac{\partial U^{i-1}}{\partial W^j} \tag{6}$$

where i is the time step, j is the time step before i. W shares the same weights across all time steps, i.e., $W^1 = W^2 = \cdots = W^T$. Where $\frac{\partial \mathcal{L}}{\partial S^i}$ represents the derivative of the loss function with respect to S_i at time step i, which can be computed using the standard chain rule, and $\frac{\partial S^i}{\partial U^i}$ can be computed using the surrogate gradient Eq. (1). $\frac{\partial U^i}{\partial U^{i-1}}$ and $\frac{\partial U^i}{\partial W^i}$ can be directly obtained from the dynamics of LIF neurons, and $\sum_{j\leq i-1}\frac{\partial U^{i-1}}{\partial W^j}$ can be computed recursively based on the Eq. (6).

B.2. Related Work

Offline RL Offline reinforcement learning (RL) trains policies entirely on static offline datasets, eliminating the need for online interactions with the environment. This paradigm is particularly valuable in scenarios where direct exploration is constrained by safety, energy, or resource limitations.

Value-based Approaches Early methods like behavior cloning (BC) [30] mapped states to actions using expert demonstrations but suffered from limited data diversity and compounding errors caused by distributional mismatch. To address these limitations, value-based methods utilizing value functions were developed and can be broadly categorized into policy constraint and regularization techniques. Policy constraints in offline RL ensure the learned policy remains close to the behavioral policy, improving reliability when extrapolating from static datasets [9, 16, 18, 25]. Methods like Batch-Constrained Q-learning (BCQ) [9] constrain actions to those observed in the dataset using perturbation models. Regularization methods [17, 19, 34] incorporate penalty terms to shape policy behaviors without explicitly estimating the behavioral policy. For example, Conservative Q-Learning (CQL) [19] mitigates overestimation and improves value function stability by regularizing value estimates toward lower bounds.

Table 1. Notation and Definitions

Notation	Description	Notation	Description
t	Time step for a LIF neuron	T	Total time steps for a LIF neuron
U^t	Membrane potential before spike emission	I^t	Input current at time step t
H^t	Membrane potential after spike emission	S^t	Spike output at time step t
γ	Decay factor for membrane potential	U_{th}	Threshold for membrane potential
U_{reset}	Reset membrane potential	Hea	Heaviside step function
Δ	Hyperparameter for surrogate gradient	l	Iteration step in RL environment
L	Total iteration steps	M	Number of Decoder Blocks
N	Context length	B	Batch Size
τ	Trajectory sequence	a_l	Action at iteration step l
D	Channel dimension	r_l	Reward at iteration step l
s_l	State at iteration step l	$\hat{R}_l$	Estimated return-to-go at iteration step l
R_l	Return-to-go at iteration step l	I_l	Model input at iteration step l
SN	Spiking neuron layer	Emb	Embedding Layer
K	Key matrix for self-attention	Q	Query matrix for self-attention
W_q	Query weight matrix	V	Value matrix for self-attention
W_v	Value weight matrix	W_k	Key weight matrix
d_k	Dimensionality of key vector	Q^t	Query vector at time step t
K^t	Key vector at time step t	V^t	Value vector at time step t
$K^\top$	Transposed key matrix	P	Positional bias matrix
$P^\top$	Transposed positional bias	P_{ij}	Bias for the i-th and j-th tokens
Q_i	i-th query vector	K_j	j-th key vector
V_j	j-th value vector	S	Local window size of PSSA
u	Mean of X	σ^2	Variance of X
ϵ	Small constant for tdBN	α	Scaling factor for membrane potential
β	Shift parameter	λ	Scale parameter
T_p	Predetermined training step	T_{cur}	Current training step
θ	Scaling factor for PTBN	$\mathcal{L}$	Loss function
A	Attention scores	sign	Signum function
D_{src}	Source dimension	D_{tgt}	Target dimension
R_m	Sum of average spike rates	$\hat{R}$	Sum of estimated spike rates

Value-free Approaches Value-free approaches do not depend on value functions. Imitation learning [13], a key example, trains policies to mimic the behavioral policy based on collected trajectories, minimizing discrepancies between the learned policy and demonstrated behavior, normally through supervised learning methods like BC. However, BC often delivers suboptimal performance on datasets with mixed-quality samples due to the tradeoff between data quantity and quality. Offline RL extends imitation learning by extracting useful transitions from suboptimal datasets through filtering [3, 6] or weighting [27, 31]. For example, some methods prioritize trajectories with higher returns or assign greater weights to advantageous state-action pairs using estimated advantages.

Conditional Sequence Modeling Recent advances in offline RL have redefined policy learning as a Conditional Sequence Modeling (CSM) problem, framing the policy generation as a supervised learning task over trajectory sequences [4, 7, 11, 14, 24]. This approach, exemplified by Decision Transformer (DT)[5], leverages transformers to model long-term dependencies by conditioning actions on past states and future returns, bypassing the need for bootstrapping. By treating trajectories as sequence prediction problems, CSM-based methods effectively capture temporal dependencies and optimize for long-term rewards without iterative updates. Recently, some studies pointed out that while CSM excels in trajectory modeling, it faces challenges with the "stitching" property, where optimal sequences are synthesized from suboptimal trajectories. They address this issue by integrating value-based regularization into CSM, as seen in recent Q-value augmented transformers [12, 32] to balance trajectory modeling with optimal ac-

tion selection. In this work, we focus on designing an SNN model within the DT framework without adding any extra value-based regularization.

Spike-driven Transformers Transformer-based Spiking Neural Networks (SNNs) have emerged as an innovative approach, combining the energy efficiency of spiking neurons with the advanced capabilities of transformers in both vision and sequence modeling tasks.

Vision Tasks Pioneering works like Spikformer [43] introduced a spiking self-attention mechanism (SSA) that employs sparse, spike-based Query, Key, and Value representations without softmax, aligning with SNN constraints. Spikformer demonstrated remarkable performance on ImageNet-1k using only 4 time steps, showcasing the potential of transformer-based SNNs. Spikingformer [40] further refined residual connections to improve energy efficiency, avoid floating-point multiplications in synaptic computing, and reduce computational complexity. Similarly, CML [41] introduced SNN-optimized downsampling to mitigate imprecise gradient backpropagation in deep SNNs, enhancing the performance of transformer-based SNNs like Spikformer on ImageNet. Spike-driven Transformers [37] proposed linear-complexity self-attention mechanisms relying on masks and addition operations, significantly reducing computational demands. Meta-SpikeFormer [38] expanded these advancements by introducing three distinct Spike-Driven Self-Attention (SDSA) modules and combining Conv-based and Transformer-based SNN blocks. This meta-architecture achieved state-of-the-art performance across diverse vision tasks, including classification, detection, and segmentation, demonstrating its versatility as a unified SNN backbone. Recent models like QKFormer [42] and SpikingResformer [26] have further optimized efficiency and scalability. QKFormer uses hierarchical Q-K self-attention to improve token and channel representations for complex tasks, while SpikingResformer integrates residual learning with dual spike attention mechanisms to efficiently scale across feature hierarchies.

Sequence Modeling Tasks SNN Transformers have shown promise in some sequential tasks like natural language processing (NLP). SpikeBERT [22] adapted Spikformer's architecture for NLP by replacing convolution modules with linear transformations and using word embeddings for token representations. SpikeGPT [44] extended SNNs to language generation, utilizing Spiking RWKV to replace self-attention, reducing computational complexity from quadratic to linear. SNN-BERT [28] introduced bidirectional spiking neurons for improved temporal modeling, bridging the gap between spiking and ANN-based models in text classification tasks. However, these methods still involve floating-point multiplications in critical components such as normalization, self-attention computations, and frameworks like knowledge distillation, limiting their adherence to pure spike-based computing principles.

C. Computational Complexity Analysis

C.1. Step-by-step Spiking Self-Attention

Given the input $X \in \mathbb{R}^{N\times D}$, where N is the sequence length and D is the channel dimension: At each time step t, the input $X^t \in \mathbb{R}^{N\times D}$ is transformed into Q, K, and V matrices in $\mathbb{R}^{N\times D}$ through three separate linear projections. The attention scores $A \in \mathbb{R}^{N\times N}$ are then calculated via matrix multiplication of Q and K, with a computational complexity of $O(DN^2)$. A masking operation is applied to prevent the model from attending to future tokens, contributing an additional computational complexity of $O(N^2)$. Subsequently, the output $Z \in \mathbb{R}^{N\times D}$ is obtained by multiplying A with V, which also has a computational complexity of $O(DN^2)$. Thus, the overall computational complexity of the SSSA mechanism for T time steps is $O(TDN^2)$.

C.2. Temporal Spiking Self-Attention

We observe that the Q, K, and V representations for all T time steps are already computed within the spiking neurons before the self-attention operation. Consequently, we can concatenate the Q, K, and V matrices along the temporal dimension and perform the self-attention operation just once, rather than T separate times. After concatenation, the dimensions of Q, K, and V become $\mathbb{R}^{N\times TD}$. Importantly, only Q, K, and V are concatenated, while X remains unchanged. This ensures that the weight matrices used to compute Q, K, and V retain their original dimensions. Correspondingly, the attention score $A \in \mathbb{R}^{N\times N}$ is calculated through the dot product of Q and K, with a computational complexity of $O(TDN^2)$. The mask operation adds a computational complexity of $O(N^2)$. Finally, the output $Z \in \mathbb{R}^{N\times TD}$ is obtained by multiplying A with V, which also has a computational complexity of $O(TDN^2)$. Thus, the overall computational complexity of the TSSA is $O(TDN^2)$.

C.3. Positional Spiking Self-Attention

Similar to TSSA, we concatenate Q, K, and V along the time dimension. To account for the Markov locality, we introduce a positional bias matrix $P \in \mathbb{R}^{N\times N}$, where each element P_{ij} represents the bias between position i and position j. A local window size S is defined, such that each row of P contains at most S non-zero elements. Instead of using matrix multiplication, we implement PSSA through element-wise multiplication. To analyze the computational complexity, we provide explanations in both vector form and its equivalent matrix form.

In the vector form operation, the computational complexity of $K_j \odot V_j$ is $O(TD)$. Similarly, the broadcast multiplication of P_{ij} and KV_j also incurs a computational complexity of $O(TD)$. During the summation process for $j = 1$ to N, it is important to note that the number of non-zero elements in P_i is at most S. Consequently, the number of multiplication and addition operations required is $O(2STD)$ and $O(STD)$, respectively. Finally, after applying element-wise multiplication with Q_i, the computational complexity for producing the final output Z_i is $O(TD)$. Therefore, for all N result vectors, the total computational complexity is $O(N(3STD + TD)) = O(STDN)$.

In the equivalent matrix form, we first compute the element-wise multiplication of the K matrix and the V matrix to generate the KV matrix. This step has a computational complexity of $O(TDN)$. Next, for each row vector of P, we perform a broadcast element-wise multiplication with the KV matrix and sum along the relevant dimension to obtain an intermediate vector. All intermediate vectors are then concatenated together. Since P is a sparse matrix, the computational complexity for this step is $O(STDN)$. Finally, we apply element-wise multiplication with the Q matrix to compute the final output matrix, which also has a computational complexity of $O(TDN)$. Thus, the overall computational complexity of PSSA is $O(STDN)$.

D. Implementation Details of PTBN

D.1. Representations of PTBN

The Progressive Threshold-dependent Batch Normalization(PTBN) we proposed implements the collaborative training of tdLN and tdBN through the following formula:

$$\mathrm{PTBN}(X) = \theta\,\mathrm{tdLN}(X) + (1-\theta)\,\mathrm{tdBN}(X) \tag{7}$$

where θ is the scaling factor for PTBN, and $X \in \mathbb{R}^{B\times N\times T\times D}$ is the input of PTBN. If we do not consider the scale and shift parameters of tdLN and tdBN, we can expand Eq. (7) as:

$$\alpha U_{\mathrm{th}}\left(\theta\frac{(X_{ijkg}-\mu_{ijk})}{\sqrt{\sigma_{ijk}^2+\epsilon}} + (1-\theta)\frac{(X_{ijkg}-\mu_g)}{\sqrt{\sigma_g^2+\epsilon}}\right) \tag{8}$$

In this formula, normalization is performed along the four dimensions of the input B, N, T, D simultaneously. Taking the MuJoCo environment as an example, at the beginning of training, we set $\theta = 1$, causing the model to primarily use tdLN, which helps it converge effectively. As training progresses, θ is gradually decreased, allowing the model to incrementally adapt to tdBN. When training reaches the specified stage T_p, θ is decreased to 0, the model fully transitions to using tdBN for the final phase of adaptation. In the inference stage, we keep $\theta = 0$, causing PTBN to degenerate into tdBN.

The parameters α and U_{th} map the input from the standard normal distribution $N(0,1)$ to the normal distribution $N(0,(\alpha U_{\mathrm{th}})^2)$. The parameter α is set to 1 in the serial network structure and set to $1/\sqrt{n}$ for a parallel network structure with n branches [39]. We change the learnable scale λ and shift β parameters of BatchNorm and LayerNorm to initialize them as αU_{th} and 0, respectively, in order to implement PTBN more concisely.

D.2. PTBN Fusion in Inference

During the inference stage, we set $\theta = 0$, causing PTBN to simplify to tdBN. As a result, PTBN can be integrated into the linear layer using the same approach as tdBN [39], as shown below:

$$W' = \lambda\frac{\alpha U_{\mathrm{th}}}{\sqrt{\sigma_D^2+\epsilon}}W \tag{9}$$

$$b' = \lambda\frac{\alpha U_{\mathrm{th}}\,(b-u_D)}{\sqrt{\sigma_D^2+\epsilon}} + \beta \tag{10}$$

Here, W and W', along with b and b', represent the weight matrix and bias term of PTBN before and after fusion, respectively. The parameters λ and β correspond to the scaling and offset factors of BatchNorm. The terms u_D and σ_D^2 denote the mean and variance of tdBN computed across the entire dataset. As previously mentioned, we can initialize αU_{th} as the learnable scale parameter λ of standard BatchNorm, thereby omitting the need for an explicit αU_{th} transformation in Eqs. (9) and (10).

E. Experimental Setup

E.1. Decision Transformer Setup

In DSFormer, the PTBN layer functions as a BatchNorm during the inference stage, relying on the alignment of input data feature dimensions. To accommodate this, we encode state, action, and return-to-go tuples as a single token. While in the original DT[5], state, return-to-go, and action are treated as three consecutive tokens during training, with the model predicting the next token based on this sequence. For a fair comparison, we also evaluate the DT variant that uses the same tokenization method as DSFormer, as provided in [29], and report the best results from the two DT models discussed in the paper. We conduct all experiments using four NVIDIA A100 Tensor Core GPUs.

E.2. Hyperparameters

The hyperparameters shared by DT and DSFormer in our experiments are summarized in Table 2. For the hidden dimension, we select the optimal value from {128, 256}. Since the human dataset and medium-replay dataset are relatively small, their batch sizes are set to 8 and 16, respectively, while the batch size for other datasets is set to 128.

Additionally, the context length is set to 32 for the human dataset and 100 for the other datasets.

DT uses GeLU as its non-linear activation function, whereas DSFormer replaces traditional activation functions with spiking neurons. Each spiking neuron is configured with a threshold U_{th} of 1.0 and a membrane potential decay factor γ of 0.25. Additionally, the spiking neuron operates over a total of $T = 4$ time steps. SpikeGPT and SpikeBERT use the same configuration as DSFormer. FCNet uses the configuration from the original paper [29].

Table 2. Hyperparameter settings.

Hyperparameter	**Value**
Number of layers	4
Hidden dimension	{128, 256}
Context length N	32, human
	100, otherwise
Batch size	8, human
	16, medium-replay
	128, otherwise
Return-to-go conditioning	1.2, expert
	1.15, medium-expert
	0.8, human, cloned
	1.0, otherwise
Optimizer	Lion
LR scheduler	get_cosine_schedule_with_warmup
Learning rate	5×10^{-3}
Epoch	50
Weight decay	10^{-4}
Gradient clip	1.0
Learning rate decay	Linear warmup for first 20% steps

E.3. Parameters

We present the number of parameters for DT and DSFormer on the halfcheetah-medium-expert dataset, as shown in Table 3. For PSSA, although the positional bias $P \in \mathbb{R}^{N\times N}$ has the shape $N \times N$, only S elements in each row are valid. We set S to 8 in all tasks, and the parameter count of PSSA during inference is nearly identical to that of DT.

Table 3. Number of parameters on halfcheetah-expert dataset. The Percentage column indicates the parameter ratio of each model relative to the DT model during the inference stage.

Model	**Training**	**Inference**	**Percentage**
DT	797575	797575	100.00%
TSSA	842399	795014	99.68%
PSSA	845599	798214	100.08%

Table 4. The FLOPs of various operations. R_m and $\hat{R}$ refer to the sum of spike firing rates across different spiking matrices.

	DT	TSSA	PSSA
Embedding	$D_{src}DN$	$D_{src}DN$	$D_{src}DN$
Q, K, V	$3D^2N$	$3TD^2NR_m$	$3TD^2NR_m$
$f(Q, K, V)$	$(2D + 3)N^2$	$TDN^2\hat{R}$	$TSDN$
Attn Linear	D^2N	TD^2NR_m	TD^2NR_m
MLP Linear1	$4D^2N$	$4TD^2NR_m$	$4TD^2NR_m$
MLP Linear2	$4D^2N$	$4TD^2NR_m$	$4TD^2NR_m$
Prediction Head	$D_{tgt}DN$	$D_{tgt}DN$	$D_{tgt}DN$

F. Theoretical Energy Evaluation

This section details the energy evaluation of DSFormer, building upon the theoretical estimation method proposed in [38].

SNNs have garnered significant attention due to their low power consumption and brain-inspired computing capabilities, which are made possible by their spiking-driven nature. In the case of a spiking-driven matrix (a matrix with elements limited to 0 or 1), element-wise multiplication is effectively a mask operation, which incurs negligible energy consumption. And matrix multiplication can be transformed into sparse addition operations and efficiently implemented on neuromorphic chips using addressable addition techniques [8]. Our energy consumption formula is as follows:

$$E_{\text{ANN}} = \text{FLOPs} \times E_{\text{MAC}} \tag{11}$$

$$E_{\text{SNN}} = \text{FLOPs} \times E_{\text{AC}} \tag{12}$$

Here, E_{MAC} represents the energy consumption of the Multiply-Accumulate (MAC) operation in ANN, and E_{AC} denotes the energy consumption of the Accumulate (AC) operation in SNN. According to [10], $E_{\text{MAC}} = 4.6\,\text{pJ}$ and $E_{\text{AC}} = 0.9\,\text{pJ}$. It is important to note that our calculation accounts for both the spiking firing rate and the total time steps of the spiking neurons, ensuring a comprehensive estimation of FLOPs. The FLOPs of our model are summarized in Table 4, while the spiking firing rates for each layer are detailed in Table 5. Additionally, the energy consumption of each model component is presented in Table 6.

In SpikeBERT, residual connections improperly introduce integer signals, which disrupt spiking characteristics and prevent matrix multiplication from being converted into sparse addition operations. Furthermore, the introduction of an additional time dimension T leads to higher energy consumption compared to the Decision Transformer (DT).

G. Ablations

SNN timestep T. We conduct ablation studies on the total time steps T of the spiking neuron in the spiking self-

Table 5. Average spiking firing rates of various spike matrices on the hopper-medium-replay dataset.

Block	Layer	TSSA	PSSA
Block 1	Q, K, V	0.33570	0.39100
	$f(Q, K, V)$	0.13363	N/A
	Attn-Linear	0.75925	0.11184
	MLP-Linear1	0.33037	0.38646
	MLP-Linear2	0.24257	0.24093
Block 2	Q, K, V	0.31161	0.36852
	$f(Q, K, V)$	0.12909	N/A
	Attn-Linear	0.80575	0.13899
	MLP-Linear1	0.31651	0.36385
	MLP-Linear2	0.19529	0.20666
Block 3	Q, K, V	0.32299	0.36977
	$f(Q, K, V)$	0.14131	N/A
	Attn-Linear	0.82538	0.18663
	MLP-Linear1	0.32341	0.35882
	MLP-Linear2	0.20368	0.24399
Block 4	Q, K, V	0.33147	0.36555
	$f(Q, K, V)$	0.13986	N/A
	Attn-Linear	0.84145	0.34599
	MLP-Linear1	0.32639	0.36521
	MLP-Linear2	0.19972	0.23693

Table 6. Energy consumption of various components of the model (μJ) on hopper-medium-replay dataset.

	DT	TSSA	PSSA
Embedding	0.9	0.9	0.9
Self-Attention	168.2	44.6	31.0
MLP	241.2	50.4	56.7
Prediction Head	0.2	0.2	0.2
Total	410.5	96.1	88.8

attention mechanism on MuJoCo-Hopper tasks. A smaller T can lead to lower energy consumption but tends to cause performance degradation, whereas a larger T increases training time and energy consumption without delivering substantial performance benefits. To balance this trade-off between energy efficiency and performance, we set $T = 4$ in our experiments. The results are shown in Table 7.

Local window size S in PSSA. We conduct ablation studies on the local window size S in the Positional Spiking Self-Attention (PSSA) on MuJoCo-Hopper tasks. As shown in Table 8, an overly small S may limit access to sufficient temporal information, while increasing the window size allows for decisions with a broader horizon but may diminish the influence of local information. We selected $S = 8$, which not only achieves the best performance but also maintains relatively low energy consumption.

Table 7. Ablation studies on SNN timestep T.

	T=1	T=2	T=4	T=8
hopper-medium-expert	109.3	109.7	110.9	109.7
hopper-medium	63.7	66.6	74.1	74.4
hopper-medium-replay	83.7	92.4	96.3	91.1
Average	85.6	89.6	93.8	91.7

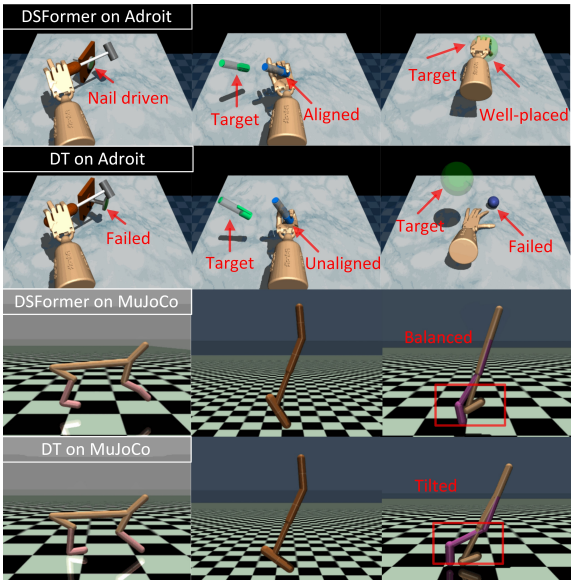
Table 8. Ablation studies on local window size in PSSA.

	S=1	S=8	S=32	S=100
hopper-medium-expert	110.4	110.9	110.7	109.7
hopper-medium	63.2	74.1	66.9	67.7
hopper-medium-replay	90.9	96.3	94.7	84.7
Average	88.1	93.8	90.8	87.4

H. Visualization

We visualize DT and DSFormer results on the medium-replay dataset in MuJoCo and the expert dataset in Adroit (Fig. S1). We will release codes for reproducing all experiments results in the paper.

Figure S1. Visualization Results: The first and third rows represent DSFormer, while the second and fourth rows represent DT.



MuJoCo Tasks	DT	WT	EDT	DC	SAN	Spike-LM	-V1	-V2	TSSA	PSSA
halfcheetah-m-e	86.8	**93.2**	48.5	91.3	85.1	75.6	91.6	91.1	91.3	91.5
walker2d-m-e	108.1	109.6	108.4	108.4	**110.6**	91.4	73.8	108.5	108.6	108.9
hopper-m-e	107.6	110.9	110.4	107.5	105.8	95.0	104.3	61.5	**111.0**	110.9
halfcheetah-m	42.6	43.0	42.5	**43.2**	42.6	42.5	42.8	42.7	42.5	42.8
walker2d-m	74.0	74.8	72.8	77.3	**81.8**	78.0	77.8	75.4	72.4	75.2
hopper-m	67.6	63.1	63.5	69.3	63.3	54.1	54.8	56.1	64.6	**74.1**
halfcheetah-m-r	36.6	39.7	37.8	40.1	**40.9**	35.9	35.0	1.4	38.7	38.8
walker2d-m-r	66.6	67.9	74.8	**76.9**	71.7	71.9	29.2	20.5	66.0	71.0
hopper-m-r	82.7	88.9	89.0	93.2	60.1	45.4	47.1	30.7	85.8	**96.3**
Average	74.7	78.7	72.0	78.6	73.4	65.5	61.8	54.2	75.7	**78.8**

Table 9. More results on MuJoCo domain.

I. More Results

We incorporated the most recent and sophisticated DT-based methods (WT [2], EDT [35], DC [15]) into our study, primarily referencing the results reported in the original papers. For DC, we reproduced the experiments using the official implementation within our experimental framework to facilitate more comprehensive comparisons. Furthermore, we included and SNN-based models (SpikeLM [36], Spike-driven Transformer V1 [37] & V2 [38] on the MuJoCo domain. For SpikeLM, we modify the official implementation to adopt a decoder-only architecture, enabling its application to sequential decision-making tasks. Additionally, we integrated the ANN-SNN hybrid model architecture, which comprises a spiking actor network (SAN [1]) and a deep critic network. To ensure compatibility with offline reinforcement learning environments, we made necessary modifications to the official implementation. All the experimental results are shown in Table 9.

References

[1] Mahmoud Akl, Deniz Ergene, Florian Walter, and Alois Knoll. Toward robust and scalable deep spiking reinforcement learning. *Frontiers in Neurorobotics*, 16:1075647, 2023. 7

[2] Anirudhan Badrinath, Yannis Flet-Berliac, Allen Nie, and Emma Brunskill. Waypoint transformer: Reinforcement learning via supervised learning with intermediate targets. *Advances in Neural Information Processing Systems*, 36: 78006–78027, 2023. 7

[3] David Brandfonbrener, William F Whitney, Rajesh Ranganath, and Joan Bruna. Quantile filtered imitation learning. *arXiv preprint arXiv:2112.00950*, 2021. 2

[4] David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. 2

[5] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 15084–15097, 2021. 2, 4

[6] Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, and Keith Ross. Bail: Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:18353–18363, 2020. 2

[7] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline RL via supervised learning? In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 2

[8] Charlotte Frenkel, David Bol, and Giacomo Indiveri. Bottom-up and top-down approaches for the design of neuromorphic processing systems: tradeoffs and synergies between natural and artificial intelligence. *Proceedings of the IEEE*, 111(6):623–652, 2023. 5

[9] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 2052–2062. PMLR, 2019. 1

[10] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014. 5

[11] Shengchao Hu, Li Shen, Ya Zhang, and Dacheng Tao. Prompt-tuning decision transformer with preference ranking. *ArXiv preprint*, abs/2305.09648, 2023. 2

[12] Shengchao Hu, Ziqing Fan, Chaoqin Huang, Li Shen, Ya Zhang, Yanfeng Wang, and Dacheng Tao. Q-value regularized transformer for offline reinforcement learning. *ArXiv preprint*, abs/2405.17098, 2024. 2

[13] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017. 2

[14] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 1273–1286, 2021. 2

[15] Jeonghye Kim, Suyoung Lee, Woojun Kim, and Youngchul Sung. Decision convformer: Local filtering in metaformer is sufficient for decision making. In *International Conference on Learning Representations*, 2024. 7

[16] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, pages 5774–5783. PMLR, 2021. 1

[17] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021. 1

[18] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11761–11771, 2019. 1

[19] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 1

[20] Yang Li, Feifei Zhao, Dongcheng Zhao, and Yi Zeng. Directly training temporal spiking neural network with sparse surrogate gradient. *Neural Networks*, 179:106499, 2024. 1

[21] Shuang Lian, Jiangrong Shen, Qianhui Liu, Ziming Wang, Rui Yan, and Huajin Tang. Learnable surrogate gradient for direct training spiking neural networks. In *IJCAI*, pages 3002–3010, 2023. 1

[22] Changze Lv, Tianlong Li, Jianhan Xu, Chenxi Gu, Zixuan Ling, Cenyuan Zhang, Xiaoqing Zheng, and Xuanjing Huang. Spikebert: A language spikformer learned from bert with knowledge distillation. 3

[23] Changze Lv, Jianhan Xu, and Xiaoqing Zheng. Spiking convolutional neural networks for text classification. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. 1

[24] Linghui Meng, Muning Wen, Chenyang Le, Xiyun Li, Dengpeng Xing, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, Yaodong Yang, et al. Offline pre-trained multi-agent decision transformer. *Machine Intelligence Research*, 20(2): 233–248, 2023. 2

[25] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019. 1

[26] Xinyu Shi, Zecheng Hao, and Zhaofei Yu. Spikingresformer: Bridging resnet and vision transformer in spiking neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5610–5619, 2024. 3

[27] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020. 2

[28] Qiaoyi Su, Shijie Mei, Xingrun Xing, Man Yao, Jiajun Zhang, Bo Xu, and Guoqi Li. Snn-bert: Training-efficient spiking neural networks for energy-efficient bert. *Neural Networks*, 180:106630, 2024. 3

[29] Hengkai Tan, Songming Liu, Kai Ma, Chengyang Ying, Xingxing Zhang, Hang Su, and Jun Zhu. Fourier controller networks for real-time decision-making in embodied learning. In *Forty-first International Conference on Machine Learning*, 2024. 4, 5

[30] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 4950–4957. ijcai.org, 2018. 1

[31] Qing Wang, Jiechao Xiong, Lei Han, Han Liu, Tong Zhang, et al. Exponentially weighted imitation learning for batched historical data. *Advances in Neural Information Processing Systems*, 31, 2018. 2

[32] Yuanfu Wang, Chao Yang, Ying Wen, Yu Liu, and Yu Qiao. Critic-guided decision transformer for offline reinforcement learning. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 15706–15714. AAAI Press, 2024. 2

[33] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018. 1

[34] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019. 1

[35] Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. *Advances in neural information processing systems*, 36:18532–18550, 2023. 7

[36] Xingrun Xing, Zheng Zhang, Ziyi Ni, Shitao Xiao, Yiming Ju, Siqi Fan, Yequan Wang, Jiajun Zhang, and Guoqi Li. Spikelm: towards general spike-driven language modeling via elastic bi-spiking mechanisms. In *Proceedings of the 41st International Conference on Machine Learning*, pages 54698–54714, 2024. 7

[37] Man Yao, Jiakui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems*

2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023. 3, 7

[38] Man Yao, JiaKui Hu, Tianxiang Hu, Yifan Xu, Zhaokun Zhou, Yonghong Tian, Bo XU, and Guoqi Li. Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips. In *The Twelfth International Conference on Learning Representations*, 2024. 3, 5, 7

[39] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 11062–11070. AAAI Press, 2021. 4

[40] Chenlin Zhou, Liutao Yu, Zhaokun Zhou, Zhengyu Ma, Han Zhang, Huihui Zhou, and Yonghong Tian. Spikingformer: Spike-driven residual learning for transformer-based spiking neural network. *ArXiv preprint*, abs/2304.11954, 2023. 3

[41] Chenlin Zhou, Han Zhang, Zhaokun Zhou, Liutao Yu, Zhengyu Ma, Huihui Zhou, Xiaopeng Fan, and Yonghong Tian. Enhancing the performance of transformer-based spiking neural networks by snn-optimized downsampling with precise gradient backpropagation. *arXiv preprint arXiv:2305.05954*, 2023. 3

[42] Chenlin Zhou, Han Zhang, Zhaokun Zhou, Liutao Yu, Liwei Huang, Xiaopeng Fan, Li Yuan, Zhengyu Ma, Huihui Zhou, and Yonghong Tian. Qkformer: Hierarchical spiking transformer using qk attention. *ArXiv preprint*, abs/2403.16552, 2024. 3

[43] Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. 3

[44] Rui-Jie Zhu, Qihang Zhao, Guoqi Li, and Jason Eshraghian. SpikeGPT: Generative pre-trained language model with spiking neural networks. *Transactions on Machine Learning Research*, 2024. 3