SPIKING GRAPH CONVOLUTIONAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Convolutional Networks (GCNs) achieve an impressive performance due to the remarkable representation ability in learning the graph information. However, GCNs, when implemented on a deep network, require expensive computation power, which makes them difficult to be deployed on battery-powered devices. In contrast, Spiking Neural Networks (SNNs), which perform a bio-fidelity inference process, offer an energy-efficient neural architecture. In this work, we propose SpikingGCN, an end-to-end framework that aims to integrate the embedding of GCNs with the biofidelity characteristics of SNNs. In particular, the original graph data are encoded into spike trains based on the incorporation of graph convolution. We further model biological information processing by utilizing a fully connected layer combined with neuron nodes. In a wide range of scenarios, including citation networks, image graph classification, and recommender systems, our experimental results show that the proposed method could gain competitive performance against state-of-art (SOTA) approaches. Furthermore, we show that SpikingGCN on a neuromorphic chip can bring a clear advantage of energy efficiency into graph data analysis, which demonstrates its great potential to construct environment-friendly machine learning models.

1 INTRODUCTION

Graph Neural Networks (GNNs), especially those using convolutional methods, have become a popular computational model for graph data analysis as the high-performance computing systems blossom during the last decade (Chen et al., 2020). One of the well-known methods in GNNs is Graph Convolutional Networks (GCNs) (Kipf & Welling, 2016), which learn a high-order approximation of a spectral graph by using convolutional layers followed by a nonlinear activation function to make the final prediction. GCNs inherit the primary characteristics from deep learning models, where the complex structure makes training and testing these networks computationally expensive, leading to signficant power consumption. It has been reported that the computation resources consumed for deep learning have grown $300,000$-fold from 2012 to 2018 (Anthony et al., 2020). The high energy consumption, when further coupled with sophisticated theoretical analysis and blurred biological interpretability of the network, has resulted in a revival of effort in developing novel energy-efficient neural architectures and physical hardware.

Inspired by the brain-like computing process, Spiking Neural Networks (SNNs) formalize the event- or clock-driven signals as inference for a set of parameters to update the neuron nodes (Brette et al., 2007). Different from conventional deep learning models that communicate information using continuous decimal values, SNNs perform inexpensive computation by transmitting the input into discrete spike trains. Such a bio-fidelity method can perform a more intuitive and simpler inference and model training than traditional networks (Maass, 1997). Another distinctive merit of SNNs is the intrinsic power efficiency on the neuromorphic hardware, which is capable of running 1 million neurons and 256 million synapses with only 70 mW energy cost (Merolla et al., 2014). Nevertheless, employing SNNs as an energy-efficient architecture to process graph data as effectively as GCNs still faces some fundamental challenges.

Challenges: (i) *Spike representation.* Despite the promising results achieved on common tasks (*e.g.,* image classification), SNN models are not trivially portable to non-Euclidean domains, such as graphs. Given the graph datasets widely used in many applications (*e.g.,* citation networks and social networks), how to extract the graph structure and transfer the graph data into spike trains poses a novel challenge. (ii) *Model generalization.* GCNs can be extended to diverse circumstances by using deeper layers. Thus, it is essential to further extend the SNNs to a wider scope of applications

where graphs are applicable. (iii) *Energy efficiency.* Except for the common metrics like accuracy or prediction loss in artificial neural networks (ANNs), the energy efficiency of SNNs on the neuromorphic chips is an important characteristic to be considered. However, neuromorphic chips are not as advanced as contemporary GPUs, and the lack of uniform standards also impacts the energy estimation on different platforms.

To tackle these fundamental challenges, we introduce Spiking Graph Neural Network (SpikingGCN): an end-to-end framework that can properly encode graphs and make a prediction for non-trivial graph datasets that arise in diverse domains. To our best knowledge, *SpikingGCN is the first-ever SNN designed for node classification in graph data*, and it can also be extended into more complex neural network structures. Overall, our main contribution is threefold: (i) We propose SpikingGCN, the first end-to-end model for node classification in SNNs, without any pre-training and conversion. The graph data is transformed into spike trains by a spike encoder. These generated spikes are used to predict the classification results. (ii) We show that the basic model inspired by GCNs can effectively merge the convolutional features into spikes and achieve competitive predictive performance. In addition, we further evaluate the performance of our model for active learning and energy efficient settings; (iii) We extend our framework to enable more complex network structures for different tasks, including image graph classification and rating predictions in recommender systems. The extensibility of the proposed model also opens the gate to perform SNN-based inference and training in various kinds of graph-based data.

2 Related Work

Spiking neural networks. The fundamental SNN architecture includes the encoder, spiking neurons, and interconnecting synapses with trainable parameters (Tavanaei et al., 2019). These procedures contribute to the substantial integrate-and-fire (IF) process in SNNs: any coming spikes lead to the change of the membrane potential in the neuron nodes; once membrane potentials reach the threshold voltage, the neuron nodes fire spikes and transmit the messages into their next nodes.

Some studies have developed the methodology along with a function to approximate the non-differentiable IF process (Jin et al., 2018; Zhang et al., 2020). Although gradient descent and error back-propagation is directly applicable for SNNs in that way, a learning phase strongly related to ANNs still causes a heavy burden on the computation. Another approach to alleviate the difficulty of training in SNNs is using an ANN-to-SNN conversion by using the pre-trained neuron weights. Kim et al. (2020) take advantage of the weights of pre-trained ANNs to construct a spiking architecture for object recognition or detection. Although those conversions can be successfully performed, multiple operators of already trained ANNs are not fully compatible with SNNs (Rueckauer et al., 2017). As a result, SNNs constructed from a fully automatic conversion of arbitrary pre-trained ANNs are not able to achieve a comparable prediction performance.

Graph neural networks. Unlike a standard neural network, GNNs need to form a state that can extract the representation of a node from its neighborhood with an arbitrary graph (Liu et al., 2020b). In particular, GNNs utilize extracted node attributes and labels in graph networks to train model parameters for a specific task, such as citation networks, social networks, protein-protein interactions (PPIs), and so on. GAT (Veličković et al., 2017) has shown that capturing the weight via an end-to-end neural network can make more important nodes receive larger weights. In order to increasingly improve the accuracy and reduce the complexity of GCNs, the extended derivatives SGC (Wu et al., 2019) eliminate the nonlinearities and collapse weight matrices between consecutive layers. FastGCN (Chen et al., 2018) successfully reduces the variance and improves the performance by sampling a designated number of nodes for each convolutional layer. Nonetheless, these convolutional GNN algorithms rely on high-performance computing systems to achieve fast inference for high-dimensional graph data due to a heavy computational cost. Since GCNs bridge the gap between spectral-based and spatial-based approaches (Xie et al., 2020), they offer desirable flexibility, extensibility, and architecture complexity. Thus, we adopt the GCN-based feature processing to construct our basic SNNs model.

3 Spiking Graph Neural Networks

Graphs are usually represented by a non-Euclidean data structure consisting of a set of nodes (vertices) and their relationships (edges). The reasoning process in the human brain depends heavily on the graph extracted from daily experience (Zhou et al., 2020). However, how to perform biologically

interpretable reasoning for the standard graph neural networks has not been adequately investigated. Thus, the proposed SpikingGCN aims to address challenges of semi-supervised node classification in a biological and energy-efficient fashion.

Graph neural networks (GNNs) conduct propagation guided by the graph structure, which is fundamentally different from existing SNN models that can only handle relatively simple image data. Instead of treating the single node as the input of an SNN model, the states of their neighborhood should also be considered. Let $\mathcal{G} = (\mathcal{V}, \boldsymbol{A})$ formally denote a graph, where $\mathcal{V}$ is the node set $\{v_1, ..., v_N\}$ and $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ represents the adjacent matrix. The entire attribute matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ includes the vectors of all nodes $[x_1, ..., x_n]^\top$. The degree matrix $\boldsymbol{D} = \text{diag}(d_1, ..., d_N)$ consists of the row-sum of the adjacent matrix $d_i = \sum_j a_{ij}$, where a_{ij} denotes the edge weight between nodes v_i and v_j. Our goal is to conduct SNN inference without neglecting the relationships between nodes.

Inference in SNN models is commonly conducted through the classic Leaky Integrate-and-Fire (LIF) mechanism (Gerstner & Kistler, 2002). Given the membrane potential V_m^t at time step t, the time constant τ_m, and the new pre-synaptic input ΔV_m, the membrane potential activity is governed by:

$$\tau_m \frac{dV_m^t}{dt} = -(V_m^t - V_{reset}) + \Delta V_m, \tag{1}$$

where V_{reset} is the signed reset voltage. The left differential item is widely used in the continuous domain, but the biological simulation in SNNs requires the implementation to be executed in a discrete and sequential way. Thus, we approximate the differential expression using an iterative version to guarantee computational availability. Updating ΔV_m using the input $I(t)$ of our network, we can formalize (1) as:

$$V_m^t = V_m^{t-1} + \frac{1}{\tau_m}(-V_m^{t-1} + V_{reset} + I(t)). \tag{2}$$

To tackle the issue of feature propagation in an SNN model, we consider a spike encoder to extract the information in the graph and output the hidden state of each node in the format of spike trains. As shown in Fig. 1, the original input graph is transformed into the spikes from a convolution perspective. To predict the labels for each node, we consider a spike decoder and treat the final spike rate as a classification result.

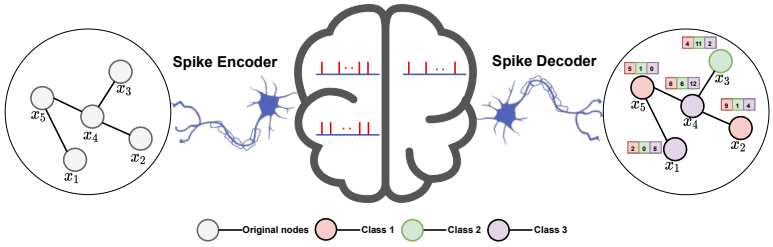


Figure 1: Schematic view of the proposed SpikingGCN. The original graph nodes will be transformed into the spike trains for the SNN model via a spike encoder. The encoder should aggregate the attributes in the graph and represent them in the format of spikes. Finally, a spike decoder, which is always made up of linear layers, may convert spike trains into various firing rates. In this case, a greater firing rate implies a higher chance that a node belongs to one of the classes.

Graph convolution. The pattern of graph data consists of two parts: topological structure and node's own features, which are stored in the adjacency and attribute matrices, respectively. Different from the general processing of images with single-channel pixel features, the topological structure will be absent if only the node attributes are considered. To avoid the performance degradation of attributes-only encoding, SpikingGCN utilizes the graph convolution method inspired by GCNs to incorporate the topological information. The idea is to use the adjacency relationship to normalize the weights, thus nodes can selectively aggregate neighbor attributes. The convolution result, *i.e.,* node representations, will serve as input to the subsequent spike encoder. Following the propagation mechanism of GCN (Kipf & Welling, 2016) and SGC (Wu et al., 2019), we form the new node representation h_i utilizing the attributes x_i of each node v_i and its local neighborhood:

$$h_i \leftarrow \frac{1}{d_i + 1} x_i + \sum_{j=1}^{N} \frac{a_{ij}}{\sqrt{(d_i + 1)(d_j + 1)}} x_j. \tag{3}$$

Here, we can express the attribute transformation over the entire graph by:

$$\boldsymbol{H} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}, \tag{4}$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacent matrix with added self-connection and $\tilde{\boldsymbol{D}}$ is the degree matrix of $\tilde{\boldsymbol{A}}$. Similar to the simplified framework as SGC, we drop the non-linear operation and focus on the convolutional process on the entire graph. As a result, (4) acts as the only convolution operation in the spike encoder. While we incorporate the feature propagation explored by GCN and SGC, we would like to further highlight our novel contributions. First, our original motivation is to leverage an SNNs-based framework to reduce the inference energy consumption of graph analysis tasks without performance degradation. GCN's effective graph Laplacian regularization approach allows us to minimize the number of trainable parameters and perform efficient inference in SNNs. Second, convolutional techniques only serve as the initial building block of SpikingGCN. More significantly, SpikingGCN is designed to accept the convolutional results in a binary form (spikes), and further detect the specific patterns among these spikes. This biological mechanism makes it suitable to be deployed on a neuromorphic chip to improve energy efficiency.

Representation encoding. The representation $\boldsymbol{H}$ consists of continuous float-point values, but SNNs accept discrete spike signals. A spike encoder is essential to take node representations as input and output spikes for the subsequent procedures. We propose to use a probability-based Bernoulli encoding scheme as the basic method to transform the node representations to the spike signals. Let $O_{i,t}^{pre} = (o_1, ..., o_d)^\top$ and λ_j denote the spikes before the fully connected layers' neurons at the t-th time step and the j-th feature in the new representation for node i, respectively. Our hypothesis is that the spiking rate should keep a positive relationship with the importance of patterns in the representations. In probability-based Bernoulli encoder, the probability p to fire a spike o_j by each feature is related to the value of λ_j in node representation as following:

$$p(o_j) \sim \text{Bernoulli}(\lambda_j), \lambda_j = \min(\lambda_j, 1.0). \tag{5}$$

Here, o_j with $j \in [d]$ denotes a pre-synaptic spike, which takes a binary value (0 or 1). Noting that λ_j derived from the convolution of neighbors is positively correlated with the feature significance. The larger the value, the greater the chance of a spike being fired by the encoder. Since the encoder generates the spike for each node on a tiny scale, we interpret the encoding module as a sampling process of the entire graph. In order to fully describe the information in the graph, we use T time steps to repeat the sampling process. It is noteworthy that the number of time steps can be defined as the resolution of the message encoded.

Charge, fire and reset in SpikingGCN. The following module includes the fully connected layer and the LIF neuron layer. Specifically, the fully connected layer takes spikes as input and outputs voltages according to trainable weights. The voltages charge LIF neurons and then conduct a series of actions, including fire spikes and reset the membrane potential.

Potential charge. General deep SNN models adopt a multi-layer network structure including linear and nonlinear counterparts to process the input (Cao et al., 2015; Hu et al., 2018). Following SGC's assumption, the depth in deep SNNs is not critical to predict unknown labels on the graph (Wu et al., 2019). Thus, we drop redundant modules except for the final linear layer (fully connected layer) to simplify our framework and increase the inference speed. We obtain the linear summation $\sum_j \psi_j o_j$ as the input of SNN structure in (2). The LIF model includes the floating-point multiplication of the constant τ_m, which is not biologically plausible. To address this challenge and avoid the additional hardware requirement when deployed on neuromorphic chips, we calculate the factor $1 - \frac{1}{\tau_m}$ as k_m and incorporate the constant $\frac{1}{\tau_m}$ into the synapse parameters ψ, then simplify the equation as:

$$V_m^t = k_m V_m^{t-1} + \sum_j \psi_j o_j. \tag{6}$$

Fire and reset. In a biological neuron, a spike is fired when the accumulated membrane potential passes a spiking threshold V_{th}. In essence, the spikes $O_{i,t}^{post}$ after the LIF neurons are generated and increase the spike rate in the output layer. We adopt the Heaviside function:

$$\mathrm{H}\left(V_m^t\right) = \begin{cases} 1 & \text{if } V_m^t \geq V_{\text{th}} \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

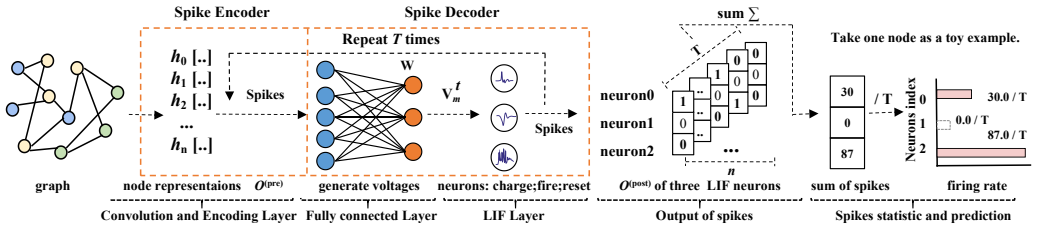


Figure 2: An illustration of SpikingGCN's detailed framework

to simulate the fundamental firing process. As shown in Fig. 2, which demonstrates our framework, T time spike trains for each node are generated from the three LIF neurons. Neurons sum the number of spikes and then divide it by T to get the firing rate of each individual. For instance, for the example nodes from ACM datasets, we get neurons' firing rates as: $fr = [30.0/T, 0.0/T, 87.0/T]^\top$, the true label: $lb = [0, 0, 1]^\top$, then the loss in training process is MSE(fr, lb). If it is in the testing phase, the predicted label would be the neuron with the max firing rate, which is 2 in this example.

Negative voltages would not trigger spikes, but these voltages contain information that (7) ignores. To compensate for the negative term, we propose to use a negative threshold to distinguish the negative characteristics of the membrane potential. To this end, we adjust the Heaviside activation function after the neuron nodes as follows:

$$\mathrm{H}\left(V_m^t\right) = \begin{cases} 1 & \text{if } V_m^t \geq V_{\text{th}}\,, \\ -1 & \text{if } V_m^t \leq -\frac{1}{\theta} V_{\text{th}}\,, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where θ is the hyperparameter that determines the negative range. In the context of biological mechanisms, we interpret the fixed activation function as an excitatory and inhibitory processes in the neurons. When capturing more information and firing spikes in response to various features, this more biologically reasonable modification also improves the performance of our model on classification tasks. The whole process is detailed by Algorithm 1 in Appendix B.

In biological neural systems, after firing a spike, the neurons tend to rest their potential and start to accumulate voltage again. We reset the membrane potential:

$$V_m^t = V_m^t - V_{th}. \tag{9}$$

Gradient surrogate. One of the most significant obstacles for SNN's training is the non-differentiable nature of activation function (7). In that case, the back-propagation algorithm can not be employed directly during the training phase. Inspired by (Roy et al., 2019), the sigmoid function is adopted to approximate the training fire phase when executing the back-propagation and stochastic gradient descent operation (Rumelhart et al., 1986). Thus, we formalize the surrogate function as follow:

$$H(V_m^t) \approx G(V_m^t) = \frac{1}{1 + \exp(-\alpha V_m^t))}, \tag{10}$$

where α can measure the approximation degree. When we train the model, the back-propagation is feasible and the gradient of the Heaviside function can be replaced as followed:

$$\frac{\partial H(V_m^t)}{\partial V_m^t} \approx \frac{\partial G(V_m^t)}{\partial V_m^t} = \frac{\alpha \cdot \exp(-\alpha V_m^t)}{(1 + \exp(-\alpha V_m^t))^2}. \tag{11}$$

4 Experiments

To evaluate the effectiveness of the proposed SpikingGCN, we conduct extensive experiments that focus on four major objectives: (i) semi-supervised node classification on citation graphs, (ii) performance evaluation under limited training data in active learning, (iii) energy efficiency evaluation on neuromorphic chips, and (iv) extensions to other application domains.

4.1 SEMI-SUPERVISED NODE CLASSIFICATION

Datasets. For node classification, we test our model on four commonly used citation network datasets: Cora, citepseer, ACM, and Pubmed (Wang et al., 2019b; Sen et al., 2008), where nodes and edges represent the papers and citation links. The statistics of the four datasets are summarized in Table 1. Sparsity refers to the number of edges divided by the square of the number of nodes.

Table 1: Statistics of the citation network datasets.

Datasets	Nodes	Edges	Attributes	Classes	Sparsity
Cora	2,708	5,429	1,433	7	0.07%
ACM	3,025	13,128	1,870	3	0.14%
citepseer	3,312	4,715	3,703	6	0.04%
Pubmed	19,717	44,324	500	3	0.01%

Baselines. We implement our proposed SpikingGCN and the following competitive baselines: GCNs (Kipf & Welling, 2016), SGC (Wu et al., 2019), FastGCN (Chen et al., 2018), GAT (Veličković et al., 2017), DAGNN (Liu et al., 2020a). We also conduct the experiments on SpikingGCN-N, a variant of SpikingGCN, which uses a refined Heaviside activation function (8) instead. For a fair comparison, we partition the data using two different ways. The first is as same as (Yang et al., 2016), which is adopted by many existing baselines in the literature. In this split method (*i.e.,* Split I), 20 instances from each class are sampled as the training datasets. In addition, 500 and 1000 instances are sampled as the validation and testing datasets respectively. For the second data split (*i.e.,* Split II), the ratio of training and testing is divided into 8:2, and 20% of training samples is further used for validation.

Table 2: Test accuracy (%) comparison of different methods. The results from the literature and our experiments are provided. The literature statistics of ACM datasets are taken from (Wang et al., 2020). The experimental scores are all averaged over 10 runs. The top 2 results are boldfaced.

Models	Cora		ACM		citepseer		Pubmed	
	Split I	Split II	Split I	Split II	Split I	Split II	Split I	Split II
GCN	81.0 ± 1.3	87.8 ± 0.5	90.0 ± 0.9	94.2 ± 0.6	69.8 ± 1.6	73.5 ± 0.5	78.4 ± 0.5	87.4 ± 0.08
SGC	81.5 ± 0.4	86.7 ± 0.8	90.5 ± 0.9	93.62 ± 0.3	71.7 ± 0.4	73.06 ± 0.2	$\mathbf{79.2 \pm 0.3}$	86.52 ± 1.6
FastGCN	80.6 ± 1.2	86.5 ± 0.6	$\mathbf{91.0 \pm 0.7}$	93.85 ± 0.5	70.0 ± 0.9	73.73 ± 0.9	77.6 ± 0.6	88.32 ± 0.2
GAT	$\mathbf{82.5 \pm 0.7}$	87.2 ± 0.5	90.9 ± 0.8	93.54 ± 0.6	71.6 ± 0.5	74.72 ± 0.7	77.5 ± 0.5	86.68 ± 0.2
DAGNN	$\mathbf{83.0 \pm 0.7}$	$\mathbf{89.70 \pm 0.1}$	$\mathbf{91.2 \pm 0.2}$	94.25 ± 0.3	$\mathbf{72.9 \pm 0.2}$	74.66 ± 0.5	$\mathbf{79.8 \pm 0.3}$	87.30 ± 0.1
SpikingGCN	80.7 ± 0.6	88.7 ± 0.5	89.5 ± 0.2	$\mathbf{94.36 \pm 0.2}$	72.5 ± 0.2	$\mathbf{77.56 \pm 0.2}$	77.6 ± 0.5	$\mathbf{89.33 \pm 0.2}$
SpikingGCN-N	81.0 ± 0.4	$\mathbf{88.7 \pm 0.1}$	90.7 ± 0.2	$\mathbf{94.78 \pm 0.2}$	$\mathbf{72.9 \pm 0.1}$	$\mathbf{77.80 \pm 0.1}$	78.5 ± 0.2	$\mathbf{89.27 \pm 0.2}$

Table 2 summarizes the node classification's accuracy comparison with the competing methods over four datasets. We show the best results we can achieve for each dataset and have the following key observations: *SpikingGCN achieves or matches SOTA results across four benchmarks on these two different dataset split methods.* It is worth noting that, when the dataset is randomly divided proportionally and SpikingGCN obtains enough data, it can even outperform the state-of-the-art approaches. For example, SpikingGCN-N outperforms DAGNN by over 3.0% on citepseer dataset. The remarkable performance of bio-fidelity SpikingGCN is attributed to three main reasons. First, as shown in Fig. 3, an appropriate T can enable our network to focus on the most relevant parts of the input representation to make a decision, similar to the attention mechanism (Veličković et al., 2017). Noting that an optimal T relies on different statistical patterns in the dataset. In another word, we can also view the Bernoulli encoder as a moderate max-pooling process on the graph features, where the salient representation of each node can have a higher probability to be the input of the network. As a result, assigning varying importance to nodes enable SpikingGCN to perform more effective prediction on the overall graph structure.

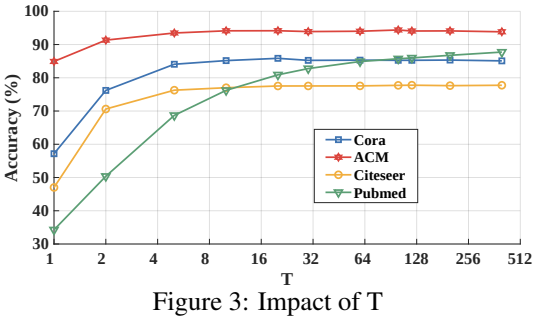


Figure 3: Impact of T

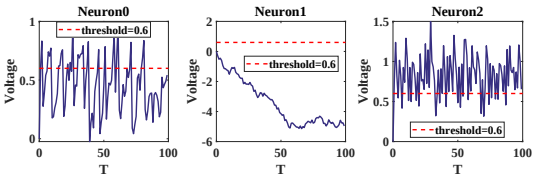


Figure 4: Membrane potential activity

Second, based on our assumption, the majority of accurate predictions benefit from attribute integration. We simplify the network and make predictions using fewer parameters, which effectively reduces the chance of overfitting. The significant performance gain indicates the better generalization ability of neural inference trained with the simplified network, which validates the effectiveness of bio-fidelity SpikingGCN. Last, the variant SpikingGCN-N has achieved better results than the original one on Cora, ACM, and citepseer datasets. As shown in Fig. 4, part of the negative voltages will be converted into negative spikes by the Heaviside activation function. The negative spikes can play a role in suppression since the spikes of T times are summed to calculate the fire ratio, which is more biologically plausible. However, the improvement seems to have no effect on Pubmed, which has the highest sparsity and the lowest number of attributes. Sparse input leads to sparse spikes and voltages, and negative spikes tend to provide overly dilute information because the hyperparameters (*e.g.,* $-1/\theta$ of Heaviside activation function) are more elusive.

4.2 SpikingGCN for Active Learning

Based on the prediction result above, we are interested in SpikingGCN's performance when the training samples vary, especially when the data is limited. Active learning has the same problem as semi-supervised learning in that labels are rare and costly to get. The objective of active learning is to discover an acquisition function that can successively pick unlabeled data in order to optimize the prediction performance of the model. Thus, instead of obtaining unlabeled data at random, active learning may help substantially increase data efficiency and reduce cost. Meanwhile, active learning also provides a way to evaluate the generalization capability of models when the data is scarce. Since SpikingGCN can achieve a 3.0 percent performance improvement with sufficient data, we are interested in how the prediction performance changes as the number of training samples increases.

Experiment Setup. We apply SpikingGCN and GCN as the active learners and observe their performance. Furthermore, three kinds of acquisition methods are considered. First, according to (Ma et al., 2013), the $\sum$- optimal (SOPT) acquisition function is model agnostic because it only depends on the graph Laplacian to determine the order of unlabeled nodes. The second one is the standard predictive entropy (PE) (Hernández-Lobato et al., 2014). Last, we consider random sampling as the baseline. Starting with only one initial sample, the accuracy is periodically reported until 50 nodes are selected. Results are reported on both Cora and ACM datasets.

Table 3: The Area under the Learning Curve (ALC) on Cora and ACM datasets.

	SOPT		PE		Random	
Datasets	GCN	SpikingGCN	GCN	SpikingGCN	GCN	SpikingGCN
Cora	71.3 ± 0.2	$\mathbf{72.9 \pm 0.3}$	59.3 ± 1.4	$\mathbf{62.6 \pm 1.1}$	57.3 ± 2.1	$\mathbf{60.8 \pm 2.0}$
ACM	85.8 ± 0.7	$\mathbf{87.7 \pm 0.4}$	83.2 ± 1.0	$\mathbf{85.1 \pm 1.3}$	82.7 ± 1.5	$\mathbf{84.7 \pm 1.7}$

The Area under the Learning Curve (ALC) [1] results are shown in Table 3. Fig 5 in Appendix C.1 provides additional details on the results. It can be seen that SOPT can choose the most informative nodes for SpikingGCN and GCN. At the same time, the PE acquisition function is a moderate strategy for performance improvement. Finally, in random strategy both models suffer from high variations during prediction as well as unstable conditions throughout the active learning process. However, no matter which strategy is adopted, SpikingGCN achieves a better generalization than GCN when the training data is scarce.

4.3 Energy Efficiency on Neuromorphic Chips

To examine the energy efficiency of SpikingGCN, we propose two metrics: i) the number of operations required to predict a node on each model, and ii) the energy consumed by SpikingGCN on neuromorphic hardware versus other models on GPUs.

The standard calculation of the computation overhead relies on operations in the hardware (Merolla et al., 2014). The operation unit of ANNs in contemporary GPUs is usually set to multiply-

[1] ALC corresponds to the area under the learning curve and is constrained to have the maximum value 1.

accumulate (MAC), and SNNs in the neuromorphic chip is synaptic operation (SOP). Furthermore, SOP is defined as the change of membrane potential (*i.e.,* voltages) in the LIF nodes, and specific statistics in the experiment refers to voltages' changes during charge and fire processes. Following the quantification methods introduced in (Hunger, 2005) and ensuring the consistency between different network constraints, we compute operations of baselines and SpikingGCN to classify one node. Table 4 shows that SpikingGCN has a significant operand advantage. According to the literature (Hu et al., 2018; Kim et al., 2020), SOPs consume far less energy than MACs, which further highlights the energy efficiency of SpikingGCN.

Table 4: Operations comparison

models	Cora	ACM	citepseer	Pubmed
GCN	67.77K	63.71K	79.54K	414.16K
SGC	10.03K	5.61K	22.22K	1.50K
FastGCN	67.54K	71.97K	141.69K	94.88K
GAT	308.94K	349.91K	499.16K	1.53M
DAGNN	281.73K	210.63K	436.11K	623.71K
SpikingGCN	1.39K	0.59K	1.19K	0.59K

Table 5: Energy consumption comparison

GCN on TITAN				
Power (W)	GFLOPS	Nodes	FLOPS	Energy (J)
280	16,310	10,000	4.14E+09	0.07
SpikingGCN on ROLLs				
Voltage (V)	Energy/spike (pJ)	Nodes	Spikes	Energy
1.8	3.7	10,000	2.73E+07	**1.01E-04**

However, the energy consumption measured by SOPs may be biased, *e.g.,* the zero spikes would also result in the voltage descending changes, which don't cost energy in neuromorphic chips. Hence calculating energy only based on operations might result in an incorrect conclusion. We further provide an alternative estimation approach as follow. Neuromorphic designs could provide event-based computation by transmitting one-bit spikes between neurons. This characteristic contributes to the energy efficiency of SNNs because they consume energy only when needed (Esser et al., 2016). For example, during the inference phase, the encoded sparse spike trains act as a low-precision synapse event, which costs the computation memory once spikes are sent from a source neuron. Considering the above hardware characteristics and the deviation of SOPs in consumption calculation, we follow the spike-based approach utilized in (Cao et al., 2015) and count the overall spikes during inference for 4 datasets, to estimate the SNN energy consumption. We list an example of energy consumption when inferring 10,000 nodes in the Pubmed dataset, as shown in Table 5.

Applying the energy consumed by each spike or operation, in Appendix C.1, we visualize the energy consumption between SpikingGCN and GNNs when employed on the recent neuromorphic chip (ROLLS (Indiveri et al., 2015)) and GPU (TITAN RTX, 24G [2]), respectively. Fig. 6 shows that SpikingGCN could use remarkably less energy than GNNs when employed on ROLLs. For example, SpikingGCN could save about 100 times energy than GCN in all datasets. Note that different from GPUs, ROLLS is firstly introduced in 2015, and higher energy efficiency of SpikingGCN can be expected in the future.

4.4 Extension to other application domains

In the above experiments, we adopt a basic encoding and decoding process, which can achieve competitive performance on the citation datasets. However, some other graph structures like image graphs and social networks can not be directly processed using graph Laplacian regularization (Belkin et al., 2006; Kipf & Welling, 2016; Wu et al., 2019). To tackle the compatibility issue, we extend our model and make it adapt to the graph embedding methods (Gilbert & Levchenko, 2004; Perozzi et al., 2014; Yang et al., 2016). Different from the graph Laplacian regularization methods like GCNs, the graph embedding methods always contain specific trainable parameters to incorporate the attributes in the graph structure. In this case, the Bernoulli encoder is unable to generate the spike trains, which perfectly represent the graph information. Taking the image graph as an example, we can see that the Bernoulli encoder cannot fully represent the pixels. Hence, the characteristics of the pixels' local Euclidean neighborhoods must be aggregated. We propose a trainable spike encoder, to allow deeper SNNs for different tasks, including classification on grid images and superpixel images, and rating prediction in recommender systems. Limited by space, we leave the implementation detail to Appendix C.2.

Result on grid images. To validate the performance of SpikingGCN on image graphs, we first apply our model to the MNIST dataset (LeCun et al., 1998). The classification results of grid images on MNIST are summarized in Table 6. We choose several SOTA algorithms including ANN and SNN

[2] https://www.nvidia.com/en-us/deep-learning-ai/products/titan-rtx/

models, which work on MNIST datasets. The depth is calculated according to the layers including trainable parameters. Since we are using a similar network structure as the Spiking CNN (Lee et al., 2016), the better result proves that our clock-driven architecture is able to capture more significant patterns in the data flow. The competitive performance of our model on image classification also proves that SpikingGCN's compatibility to different graph scenarios.

Table 6: Test accuracy (%) comparison on MNIST dataset. The best results are boldfaced.

Models	Type	Depth	Accuracy
SplineCNN (Fey et al., 2018)	ANN	8	99.22
LeNet5 (LeCun et al., 1998)	ANN	4	99.33
LISNN (Cheng et al., 2020)	SNN	6	99.50
Spiking CNN (Lee et al., 2016)	SNN	4	99.31
S-ResNet (Hu et al., 2018)	SNN	8	**99.59**
SpikingGCN (Ours)	SNN	4	99.35

Results on superpixel images. We select the MNIST superpixel dataset (Monti et al., 2017a) for the comparison with the grid experiment mentioned above. The results of the MNIST superpixel experiments are presented in Table 7. Since our goal is to prove the generalization of our model on different scenarios, we only use 20 time steps to conduct this subgraph classification task and achieve the mean accuracy of 94.50% over 10 runs. It can be seen that SpikingGCN is readily compatible with the different convolutional methods of the graph and obtain a competitive performance through a biological mechanism.

Table 7: Test accuracy comparison on MNIST dataset. The best results are boldfaced. Baseline numbers are taken from Fey et al. (2018).

Models	Accuracy
ChebNet Defferrard et al. (2016)	75.62
MoNet Monti et al. (2017a)	91.11
SplineCNN Fey et al. (2018)	95.22
SpikingGCN (Ours)	94.50

Table 8: Test RMSE scores with MovieLens 100K datasets. Baselines numbers are taken from (van den Berg et al., 2017).

Models	RMSE Score
MC (Candès & Recht, 2012)	0.973
GMC (Kalofolias et al., 2014)	0.996
GRALS (Rao et al., 2015)	0.945
sRGCNN (Monti et al., 2017b)	0.929
GC-MC (van den Berg et al., 2017)	0.910
SpikingGCN (Ours)	0.924

Results on recommender systems. We also evaluate our model with a rating matrix extracted from MovieLens 100K [3] and report the RMSE scores compared with other matrix completion baselines, as shown in Table 8. The comparable loss 0.924 indicates that our proposed framework can also be employed in recommender systems. Because the purpose of this experiment is to demonstrate the applicability of SpikingGCN in recommender systems, we have not gone into depth on the design of a specific spike encoder. We leave this design in the future work since it is not the focus of the current paper.

5 Conclusions

In this paper, we present SpikingGCN, a first-ever bio-fidelity and energy-efficient framework focusing on graph-structured data, which encodes the node representation and makes the prediction with less energy consumption. In our basic model for citation networks, the encoded spike trains are processed by a simple linear layer combined with a neuron layer. We conduct extensive experiments on node classification with four public datasets, including Cora, citepseer, ACM, and Pubmed. Compared with other SOTA approaches, we demonstrate that SpikingGCN achieves the best accuracy with the lowest computation cost, which leads to much-reduced energy consumption. Furthermore, SpikingGCN also exhibits great generalization when confronted with limited data. In our extended model for more graph scenarios, SpikingGCN also has the potential to compete with the SOTA models on tasks from computer vision or recommender systems. Relevant results and discussions are presented to offer key insights on the working principle, which may stimulate future research on environmentally friendly and biological algorithms.

[3] https://grouplens.org/datasets/movielens/

REFERENCES

Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *CoRR*, abs/2007.03051, 2020.

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *J. Mach. Learn. Res.*, 7:2399–2434, 2006.

Romain Brette, Michelle Rudolph, Ted Carnevale, et al. Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.*, 23(3):349–398, 2007.

Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119, 2012.

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016.

Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *In IJCV*, 113(1):54–66, 2015.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv:1801.10247*, 2018.

Liang Chen, Jintang Li, Jiaying Peng, Tao Xie, Zengxu Cao, Kun Xu, Xiangnan He, and Zibin Zheng. A survey of adversarial learning on graphs. *arXiv:2003.05730*, 2020.

Xiang Cheng, Yunzhe Hao, Jiaming Xu, and Bo Xu. LISNN: improving spiking neural networks with lateral interactions for robust object recognition. In *IJCAI*, pp. 1519–1525. ijcai.org, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pp. 3837–3845, 2016.

Peter U. Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers Comput. Neurosci.*, 9:99, 2015.

Steven K Esser, Paul A Merolla, John V Arthur, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the national academy of sciences*, 113(41): 11441–11446, 2016.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *CVPR*, pp. 869–877. IEEE Computer Society, 2018.

Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

Anna C Gilbert and Kirill Levchenko. Compressing network graphs. In *Proceedings of the LinkKDD workshop at the 10th ACM Conference on KDD*, volume 124, 2004.

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, pp. 639–648. ACM, 2020.

José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *NIPS*, pp. 918–926, 2014.

Yangfan Hu, Huajin Tang, Yueming Wang, and Gang Pan. Spiking deep residual network. *arXiv:1805.01352*, 2018.

Raphael Hunger. *Floating point operations in matrix-vector calculus*. 2005.

Giacomo Indiveri, Federico Corradi, and Ning Qiao. Neuromorphic architectures for spiking deep neural networks. In *IEDM*, pp. 4–2, 2015.

Yingyezhe Jin, Wenrui Zhang, and Peng Li. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *NIPS*, 2018.

Vassilis Kalofolias, Xavier Bresson, Michael M. Bronstein, and Pierre Vandergheynst. Matrix completion on graphs. *CoRR*, abs/1408.1717, 2014.

Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *AAAI*, pp. 11270–11277, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in neuroscience*, 12:435, 2018.

Junhaeng Lee, Tobi Delbrück, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *CoRR*, abs/1608.08782, 2016.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *SIGKDD*, pp. 338–348, 2020a.

Yang Liu, Jiaying Peng, Liang Chen, and Zibin Zheng. Abstract interpretation based robustness certification for graph convolutional networks. In *24th ECAI*, 2020b.

Yifei Ma, Roman Garnett, and Jeff G. Schneider. Σ-optimality for active learning on gaussian random fields. In *NIPS*, pp. 2751–2759, 2013.

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

Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*, pp. 5425–5434. IEEE Computer Society, 2017a.

Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *NIPS*, pp. 3697–3707, 2017b.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *KDD*, pp. 701–710. ACM, 2014.

Nikhil Rao, Hsiang-Fu Yu, Pradeep Ravikumar, and Inderjit S. Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *NIPS*, pp. 2107–2115, 2015.

Deboleena Roy, Indranil Chakraborty, and Kaushik Roy. Scaling deep spiking neural networks with binary stochastic activations. In *ICCC*, pp. 50–58, 2019.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.*, 11:682, 2017.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *ACL (1)*, pp. 3645–3650. Association for Computational Linguistics, 2019.

Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019.

Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion. *CoRR*, abs/1706.02263, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, et al. Graph attention networks. *arXiv:1710.10903*, 2017.

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, pp. 165–174. ACM, 2019a.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *WWW*, pp. 2022–2032, 2019b.

Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. AM-GCN: adaptive multi-channel graph convolutional networks. In *KDD*, pp. 1243–1253. ACM, 2020.

Felix Wu, Amauri H Souza Jr, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Fenfang Xie, Zengxu Cao, Yangjun Xu, Liang Chen, and Zibin Zheng. Graph neural network and multi-view learning based mobile application recommendation in heterogeneous graphs. In *2020 IEEE (SCC)*, 2020.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 40–48. JMLR.org, 2016.

Malu Zhang, Jiadong Wang, Zhixuan Zhang, et al. Spike-timing-dependent back propagation in deep spiking neural networks. *arXiv preprint arXiv:2003.11837*, 2020.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Appendix

Organization of the Appendix. In this Appendix, we first discuss some additional related work that provides a more complete context of the proposed SpikingGCN model. We then describe the detailed training process of the model, which is accompanied by the link to access the entire source code. Finally, we show additional experimental results that complement the ones presented in the main paper.

A Additional Related work

Spiking Neural Networks One popular way to build the SNNs models is the spike-timing-dependent-plasticity (STDP) learning rule, where the synaptic weight is adjusted according to the interval between the pre- and postsynaptic spikes. Diehl & Cook (2015) propose an unsupervise learning model, which utilizes more biologically plausible components like conductance-based synapses and different STDP rules to achieve competitive performance on the MNIST dataset. Lee et al. (2018) introduce a pre-training scheme using biologically plausible unsupervised learning to better initialize the parameters in multi-layer systems. Although STDP models provide a closer match to biology for the learning process, how to achieve a higher level function like classification using supervised learning is still unsolved (Cao et al., 2015). Besides, it can easily suffer from prediction performance degradation compared with supervised learning models.

Energy consumption estimation. An intuitive measurement of the model's energy consumption is investigating the practical electrical consumption. Strubell et al. (2019) propose to repeatedly query the NVIDIA System Management Interface [4] to obtain the average energy consumption for training deep neural networks for natural language processing (NLP) tasks. Canziani et al. (2016) measure the average power draw required during inference on GPUs by using the Keysight 1146B Hall effect current probe. However, querying the practical energy consumption requires very strict environment control (e.g., platform version and temperature), and might include the consumption of background program, which results in the inaccuracy measurement. Another promising approach to estimate the model's energy consumption is according to the operations during training or inference. Anthony et al. (2020) develop a tool for calculating the operations of different neural network layers, which helps to track and predict the energy and carbon footprint of ANN models. Some SNN approaches (Hu et al., 2018; Kim et al., 2020) successfully access the energy consumed by ANN and SNN models by measuring corresponding operations multiplied by theoretical unit power consumption. This kind of methods can estimate the ideal energy consumption excluding environmental disturbance. In addition, contemporary GPU platforms are much more mature than SNN platforms or neuromorphic chips (Merolla et al., 2014; Indiveri et al., 2015). As a result, due to the technical restriction of employing SpikingGCN on neuromorphic chips, we theoretically estimate the energy consumption in the experimental section.

B Algorithm and source code

Algorithm 1 shows the detailed training process of the proposed SpikingGCN model. The source code can be accessed via `https://anonymous.4open.science/r/SpikingGCN-1527`.

C Additional Experimental Results

We report additional experimental results that complement the ones reported in the main paper.

C.1 Active Learning Curves and Energy Efficiency Experiments

We provide the active learning curves of SpikingGCN and GCN in Fig. 5, which are consistent with the statistics reported in Table 3. Fig. 6 shows the remarkable energy difference between SpikingGCN and GNN based models. First, the sparse characteristic of graph datasets fits the spike-based encoding method. Furthermore, the zero values in node representations would have no chance to inspire a synapse event (spike) on a neuromorphic chip, leading to no energy consumption. Second, our simplified network architecture only contains two main neuron layers: a single fully connected layer and an LIF layer. Consider Pubmed as an example. Few attributes and a sparse adjacency matrix result in sparse spikes, and the smaller number (*i.e.,* 3) of classes also require fewer neurons.

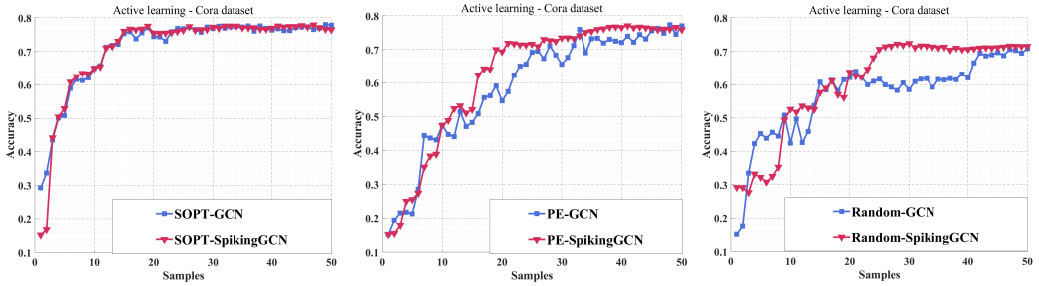
[4] nvidia-smi: `https://bit.ly/30sGEbi`

Algorithm 1 Model Training of SpikingGCN

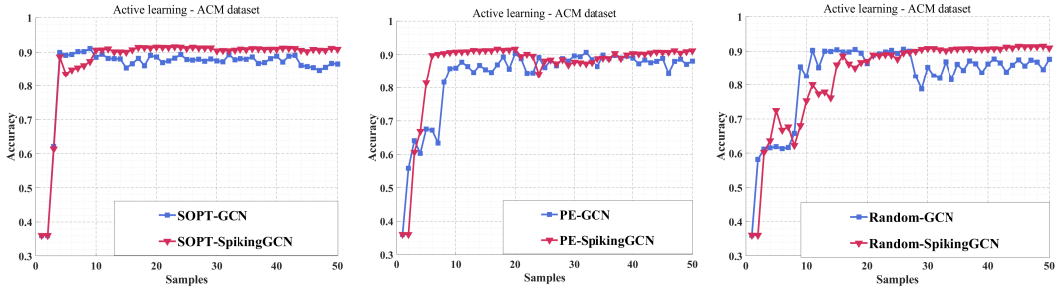
Input: Graph $\mathcal{G}(\mathcal{V}, \mathbf{A})$; input attributes $x_i \in \mathbf{X}$; one-hot matrix of label $y_i \in \mathbf{Y}_o$;
Parameter: Learning rate β; Weight matrix $\mathbf{w}$; embedding function EMBEDDING(); encoding function ENCODING();
charge, fire, reset functions CHARGE(), FIRE(), RESET()
Output: Firing rate vector $\hat{y}_i$ for training subset $\mathcal{V}_o$, which is the prediction

while not converge **do**
  Sample a mini-batch nodes $\mathcal{V}_l$ from the training nodes $\mathcal{V}_o$
  for each node $i \in \mathcal{V}_l$ **do**
    $h_i \leftarrow \text{EMBEDDING}(\mathbf{A}, x_i)$ // Eq. (3)(4)
    for $t = 1...T$ **do**
      $O_{i,t}^{pre} \leftarrow \text{ENCODING}(h_i)$ // Eq. (5)
      $V_m^t = \text{CHARGE}(\mathbf{W} \cdot O_{i,t}^{pre})$ // Eq. (2)
      $O_{i,t}^{post} = \text{FIRE}(V_m^t)$
      $V_m^t = \text{RESET}(V_m^t)$ // Eq. (9)
    end for
    $\hat{y}_i \leftarrow \frac{1}{T} \sum O_{i,t}^{post}$
  end for
  Perform meta update, $\mathbf{w} \leftarrow \mathbf{w} - \beta \nabla_{\mathbf{w}} \mathcal{L}(y_i, \hat{y}_i)$
end while

This promising results imply that SpikingGCN could have the potential to achieve more significant advantages in energy consumption than general GNNs.



(a) Active learning on the Cora dataset



(b) Active learning on the ACM dataset

Figure 5: Active learning curves for both Cora and ACM datasets.

C.2 SPIKINGGCN ON OTHER APPLICATION DOMAINS

Results on image grids. The MNIST dataset contains 60,000 training samples and 10,000 testing samples of handwritten digits from 10 classes. Each image has $28 \times 28 = 784$ grids or pixels, hence we treat each image as a node which has 784 features. It is worth noting that the grid image classification is identical to the citation networks where node classes will be identified, with the exception of the absence of an adjacent matrix. To extend our model, we adopt the traditional

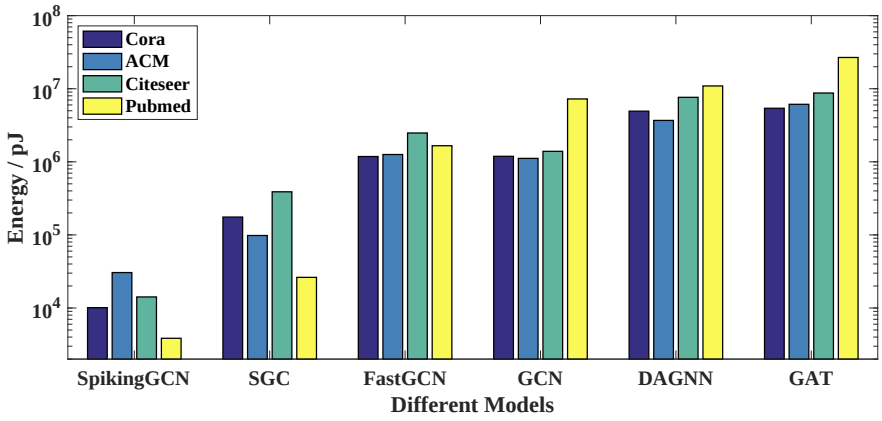


Figure 6: The energy consumption of SpikingGCN and baselines on their respective hardware.

convolutional layers and provide the trainable spike encoder for graph embedding models, and the extended framework is given by Fig. 7. Since the LIF neuron models contain the leaky parameters τ_m, which can decay the membrane potential V_m^t and activate the spikes on a small scale, we adopt the Integrate-and-Fire (IF) process to maintain a suitable firing rate for the encoder. The membrane activity happening in the spike encoder can be formalized as:

$$V_m^t = V_m^{t-1}(1 - |\mathrm{H}\left(V_m^{t-1}\right)|) + X(t), \tag{12}$$

where $X(t)$ is the convolutional output at time step t, and $\mathrm{H}\left(V_m^{t-1}\right)$ is given in (8). As shown in Fig. 7, the convolutional layers combined with the IF neurons will perform an auto-encoder function for the input graph data. After processing the spike trains, the fully connected layers combined with the LIF neurons can generate the spike rates for each class, and we will obtain the prediction result from the most active neurons.

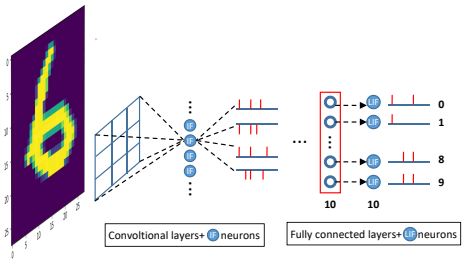


Figure 7: An extended model for deep SNNs

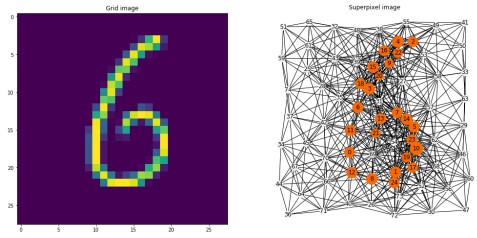
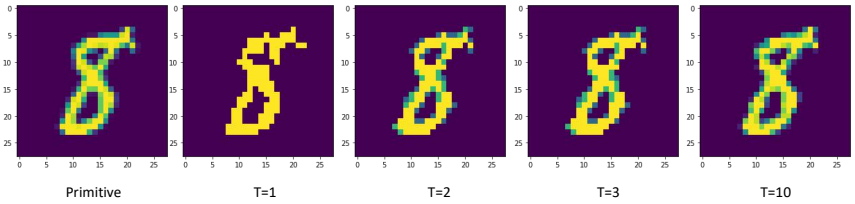


Figure 8: Comparison between grid and superpixel images

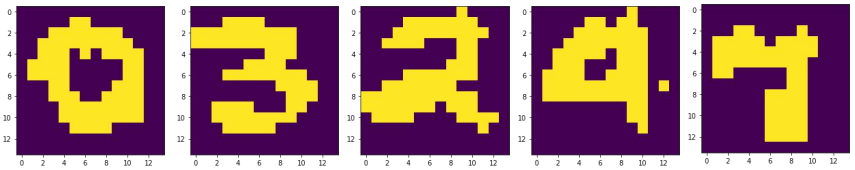
Results on superpixel images. Another more complex graph structure is the superpixel images. Compared with the general grid images, superpixel images represent each picture as a graph which consists of connected nodes. Hence the classification task is defined as the prediction on the subgraphs. Another important distinction is that the superpixel images require to construct the connectivity between chosen nodes. A comparison between the grid and superpixel images is shown in Fig. 8, where 75 superpixels are processed as the representation of the image.

One of the important steps when processing the superpixel data is learning effective graph embedding extracted from the graph. To demonstrate the ability of our model when predicting based on the superpixel images, we empirically follow the convolutional approach utilized in SplineCNN (Fey et al., 2018) to further aggregate the connectivity of superpixels. The trainable kernel function based on B-splines can make the most use of the local information in the graph and filter the input into a representative embedding. Similar to the framework proposed in Fig 7, the experiments on superpixel images also follow the structures as grid image experiments, where the convolutional layers & IF neurons enable the spike representations, and the fully connected layers & LIF neurons are responsible for the classification results.

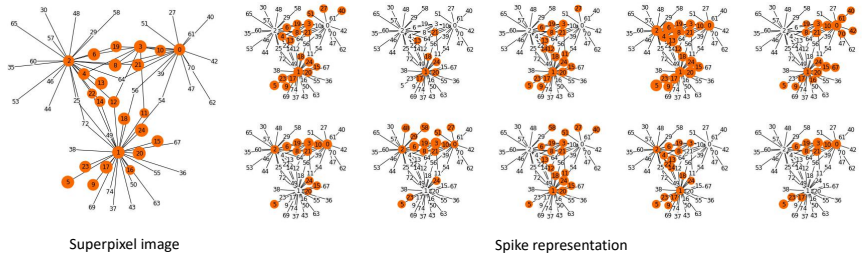
In addition, we also provide a unique perspective to understand the mechanism of our model. In particular, our spike encoder can be regarded as a sampling process using spike train representation. The scenario of the image graph provides us an ideal chance to visualize the data processing in our model. Regarding the experiments of grid and superpixel images, we extract the outputs of our spike encoder and visualize them in Fig. 9 (c), along with other observations. First, the Bernoulli encoder mentioned above can be viewed as a sampling process with respect to the pixel values. As the time step increases, the encoder almost rebuilds the original input. However, the static spike encoder can not capture more useful features from the input data. Thus, our trainable encoder performs the convolution procedure and stimulates the IF neurons to fire a spike. As shown in Fig. 9 (b) and (c), by learning the convolutional parameters in the encoder, the spike encoder successfully detects the structure patterns and represents them in a discrete format.



(a) Outputs of Bernoulli encoder in grid images



(b) Outputs of trainable encoder in grid images



(c) Outputs of trainable encoder in superpixel images

Figure 9: Visualization of the spike trains generated by the spike encoder. We extract these features from the MNIST dataset for demonstration. Grid images: (a) shows the spike trains from a simple Bernoulli encoder, and we list the different time steps which indicate different precision. (b) depicts the spikes from the trainable spike encoder, in which the overall shape patterns are learned. Superpixel images: (c) demonstrates the spikes from the trainable encoder, and the encoding results indicate the successful detection of local aggregation.

Spike encoder for recommender systems. Much research has tried to leverage the graph-based methods in analyzing social networks (van den Berg et al., 2017; Wang et al., 2019a; He et al., 2020). To this end, we extend our framework to the recommender systems, where users and items form a bipartite interaction graph for message passing. We tackle the rating prediction in recommender systems as a link classification problem. Starting with MovieLens 100K datasets, we take the rating pairs between users and items as the input, transform them into suitable spike representations, and finally output the classification class via firing rate. To effectively model this graph-structured data, we build our trainable spike encoder based on the convolutional method used in GC-MC (van den Berg et al., 2017). In particular, GC-MC applies a simple but effective convolutional approach based on differentiable message passing on the bipartite interaction graph, and reconstruct the link utilizing a bilinear decoder.