Spiking Graph Convolutional Networks

Zulun Zhu^{1,2*} , **Jiaying Peng**^{1} , **Jintang Li** ^{1} , **Liang Chen**^{1†} , **Qi Yu** ^{2} and **Siqiang Luo** ^{3}
^{1}Sun Yat-Sen University
^{2}Rochester Institute of Technology
^{3}Nanyang Technological University
zulun.zhu@gmail.com, {pengjy36,lijt55}@mail2.sysu.edu.cn, chenliang6@mail.sysu.edu.cn, qi.yu@rit.edu, siqiang.luo@ntu.edu.sg

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, making 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. 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 (*e.g.,* citation networks, image graph classification, and recommender systems), our experimental results show that the proposed method could gain competitive performance against state-of-the-art 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. One of well-known methods in GNNs is Graph Convolutional Networks (GCNs) [Kipf and 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. Like most of the deep learning models, GCNs incorporate complex structures with costly training and testing process, leading to significant 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 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 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-

[*] This work was done when the first author was with Rochester Institute of Technology.
[†] Corresponding Author

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. The code and Appendix are available on Github[1].

2 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. As this work refers to the methods in GNNs and SNNs, we list the frequently used notations in Table 8 in Appendix.

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 $\mathscr{G} = (\mathscr{V}, \boldsymbol{A})$ formally denote a graph, where $\mathscr{V}$ is the node set $\{v_1, \dots, v_N\}$ and $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ represents the adjacent matrix. Here N is the number of nodes. The entire attribute matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ includes the vectors of all nodes $[x_1, \dots, x_N]^\top$. The degree matrix $\boldsymbol{D} = \mathrm{diag}(d_1, \dots, d_N)$ 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. Each node has d dimensions. 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 and 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:

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

[1]https://github.com/ZulunZhu/SpikingGCN.git

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.

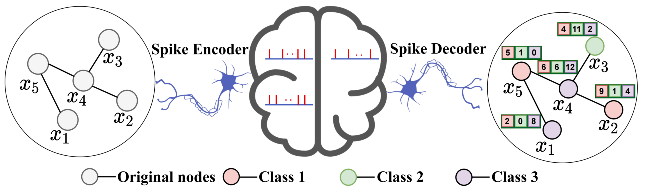

Figure 1: Schematic view of the proposed SpikingGCN.

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 and 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:

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

where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacent matrix with added self-connection, K is the graph convolution layer number 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 $O_{i,t}^{pre} = (o_{i1}, ..., o_{id})$ and λ_{ij} 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_{ij} by each feature is related to the value of λ_{ij} in node representation as following:

$$p(o_{ij}) \sim \text{Bernoulli}(\lambda_{ij}), \lambda_{ij} = \min(\lambda_{ij}, 1.0). \tag{5}$$

Here, o_{ij} with $j \in [d]$ denotes a pre-synaptic spike, which takes a binary value (0 or 1). Note that λ_{ij} 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. 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]. 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 $\sum_j \psi_j o_{ij}$ 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:

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

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:

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

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, *i.e.,* 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. Inspired by [Kim *et al.*, 2020], 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:

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

Model Feasibility Analysis. Since the input spikes can be viewed as a rough approximation of original convolutional results in the initial graph, two key questions remain: (i) does this proposed method really work for the prediction task? (ii) how to control the information reduction of the sampled spikes compared with other GNN models? It turns out that although equation (6) shows a rough intuitive approximation

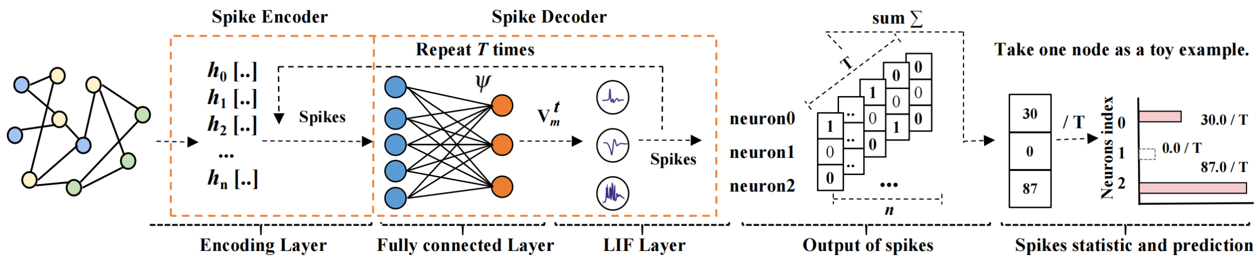

Figure 2: An illustration of SpikingGCN's detailed framework

of the graph input using trainable linear combination, it cannot fully explain why SpikingGCN can achieve comparable performance with other real-value GNN models. Next, we will show that our spike representations $\psi_j o_j$ is very close to the real-value output with a high probability.

To explain why SpikingGCN can provide an accurate output using spike trains, let us list SGC ([Wu *et al.*, 2019]) as the model for comparison. SGC adopts a similar framework to our model, which transfers the convolution result $\boldsymbol{H}$ into a fully connected layer in a real-value format. Given the parameters $\psi = (\psi_1, ...\psi_j, ..., \psi_d)$, the real-value convolution result $h = (\lambda_1, ..., \lambda_j, ..., \lambda_d)$ and the spike representations $O_{i,t}^{pre} = (o_1, ..., o_j, ..., o_d)$ for one node, we have:

$$\Pr(o_j = 1) = \lambda_j, \quad \Pr(o_j = 0) = 1 - \lambda_j \tag{10}$$

Note that firing a spike in each dimension of a node is independent. When merging our case into a generalization of the Chernoff inequalities for binomial distribution in [Chung and Lu, 2002], we derive the following estimation error bound.

Let $o_1, ..., o_j, ..., o_d$ be independent random variables with equation (10). For $Z = \sum_{j=1}^{d} \psi_j o_j$, we have $E(Z) = \sum_{j=1}^{d} \psi_j \lambda_j$ and we define $\sigma = \sum_{j=1}^{d} \psi_j^2 \lambda_j, \sigma' = \sum_{j=1}^{d} \psi_j^2$. Then we have

$$\Pr(Z < E(Z) - \varepsilon) \le e^{-\varepsilon^2/2\sigma} \le e^{-\varepsilon^2/2\sigma'} \tag{11}$$

$$\Pr(Z > E(Z) + \varepsilon) \le e^{-\frac{\varepsilon^2}{2(\sigma + \hat{\psi}\varepsilon/3)}} \le e^{-\frac{\varepsilon^2}{2(\sigma' + \hat{\psi}\varepsilon/3)}} \tag{12}$$

where $\hat{\psi} = max\{\psi_1, ...\psi_j, ..., \psi_d\}$. Note that $E(Z)$ is exactly the output of the SGC's trainable layer, and Z is the output of SpikingGCN after a linear layer. By applying the upper and lower bounds, the failure probability will be at most $p_f = e^{-\frac{\varepsilon^2}{2(\sigma' + \hat{\psi}\varepsilon/3)}}$. For question (i) mentioned earlier, we guarantee that our spike representations have approximation to SGC's output with at least $1 - p_f$ probability. For question (ii), we also employ the L_2 regularization and parameter clip operations during our experiments to control the σ' and $\hat{\psi}$ respectively, which can further help us optimize the upper and lower bounds.

3 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. Due to the limitation of space, we leave the active learning experiments in Appendix C.2.

3.1 Semi-Supervised Node Classification

Datasets. For node classification, we test our model on four commonly used citation network datasets: Cora, citeseer, ACM, and Pubmed [Wang *et al.*, 2019], 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.

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%
citeseer	3,312	4,715	3,703	6	0.04%
Pubmed	19,717	44,324	500	3	0.01%

Table 1: Statistics of the citation network datasets.

Baselines. We implement our proposed SpikingGCN and the following competitive baselines: GCNs [Kipf and Welling, 2016], SGC [Wu *et al.*, 2019], FastGCN [Chen *et al.*, 2018], GAT [Veličković *et al.*, 2017], DAGNN [Liu *et al.*, 2020]. 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 the 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 to testing is 8:2, and 20% of training samples is further used for validation.

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 citeseer dataset. The detailed discussion of the performance can be found in Appendix C.1.

	Cora		ACM		citeseer		Pubmed	
Models	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	**79.2 ± 0.3**	86.52 ± 1.6
FastGCN	80.6 ± 1.2	86.5 ± 0.6	**91.0 ± 0.7**	93.85 ± 0.5	70.0 ± 0.9	73.73 ± 0.9	77.6 ± 0.6	88.32 ± 0.2
GAT	**82.5 ± 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	**84.2 ± 0.7**	**89.70 ± 0.1**	**91.2 ± 0.2**	94.25 ± 0.3	**72.9 ± 0.2**	74.66 ± 0.5	**79.8 ± 0.3**	87.30 ± 0.1
SpikingGCN	80.7 ± 0.6	88.7 ± 0.5	89.5 ± 0.2	**94.36 ± 0.2**	72.5 ± 0.2	**77.56 ± 0.2**	77.6 ± 0.5	**89.33 ± 0.2**
SpikingGCN-N	81.0 ± 0.4	**88.7 ± 0.1**	90.7 ± 0.2	**94.78 ± 0.2**	**72.9 ± 0.1**	**77.80 ± 0.1**	78.5 ± 0.2	**89.27 ± 0.2**

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]. All results are averaged over 10 runs. The top 2 results are boldfaced.

3.2 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. In this experiment, only the basic SpikingGCN is conducted to evaluate the energy efficiency. The reason we omit SpikingGCN-N is that the negative spikes cannot be implemented on neuromorphic hardware.

We note that training SNN models directly on neuromorphic chip is rarely explored ([Thiele *et al.*, 2019]). In that case, we employ the training phase on GPUs and estimate the energy consumption of test phase on neuromorphic hardware. More importantly, a specific feature of semi-supervised on GNNs is that test data is also visible during the training process. Therefore, the convolutional part during the training covers the global graph. Then during the test phase, no MAC operation is required by our SNN model because all of the data has been processed on GPUs.

Estimating 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-accumulate (MAC), and for SNNs in the neuromorphic chip is the 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 refer 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 the operations of baselines and SpikingGCN to classify one node. Table 3 shows that SpikingGCN has a significant operand reduction. 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.

However, the energy consumption measured by SOPs may be biased, *e.g.,* the zero spikes would also result in the voltage descending changes, which does not require new energy consumption in neuromorphic chips [Indiveri *et al.*, 2015]. Hence, calculating energy cost only based on operations may result in an incorrect conclusion. To address this issue, 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 4.

models	Cora	ACM	citeseer	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 3: Operations 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**

Table 4: Energy consumption comparison

Applying the energy consumed by each spike or operation, in Appendix C.3, 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.

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

3.3 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 (*i.e.,* [Kipf and Welling, 2016; Wu *et al.*, 2019]). To tackle the compatibility issue, we extend our model and make it adapt to the graph embedding methods (*i.e.,* [Yang *et al.*, 2016]). 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.4.

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 5. We choose several SOTA algorithms including ANN and SNN 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.

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

Table 5: Test accuracy (%) comparison on MNIST. The best results are boldfaced.

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 6. 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.

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 6: Test accuracy comparison on MNIST. The best results are boldfaced. Baseline numbers are taken from [Fey *et al.*, 2018].

Models	RMSE Score
MC [Candès and 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

Table 7: Test RMSE scores with MovieLens 100K datasets. Baselines numbers are taken from [van den Berg *et al.*, 2017].

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 in Table 7. 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.

4 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, we conduct extensive experiments on node classification with four public datasets. Compared with other SOTA approaches, we demonstrate that SpikingGCN achieves the best accuracy with the lowest computation cost and 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.

Acknowledgments

The research is supported by the Key-Area Research and Development Program of Guangdong Province (2020B010165003), the Guangdong Basic and Applied Basic Research Foundation (No. 2020A1515010831), the Guangzhou Basic and Applied Basic Research Foundation (No. 202102020881), the Tencent AI Lab RBFR2022017, and the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (No. 2017ZT07X355). Qi Yu is supported in part by an NSF IIS award IIS-1814450 and an ONR award N00014-18-1-2875. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agency.

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

References

[Anthony *et al.*, 2020] 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.

[Brette *et al.*, 2007] 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.

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

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

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

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

[Chung and Lu, 2002] Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002.

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

[Esser *et al.*, 2016] 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.

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

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

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

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

[Indiveri *et al.*, 2015] Giacomo Indiveri, Federico Corradi, and Ning Qiao. Neuromorphic architectures for spiking deep neural networks. In *IEDM*, pages 4–2, 2015.

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

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

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

[LeCun *et al.*, 1998] 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.

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

[Liu *et al.*, 2020] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *SIGKDD*, pages 338–348, 2020.

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

[Merolla *et al.*, 2014] 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.

[Monti *et al.*, 2017a] 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*, pages 5425–5434. IEEE Computer Society, 2017.

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

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

[Thiele *et al.*, 2019] Johannes Christian Thiele, Olivier Bichler, and Antoine Dupret. Spikegrad: An ann-equivalent computation model for implementing backpropagation with spikes. *arXiv preprint arXiv:1906.00851*, 2019.

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

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

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

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

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

[Yang *et al.*, 2016] 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*, pages 40–48. JMLR.org, 2016.

[Zhou *et al.*, 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.