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 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 are 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. We leave further related work in Appendix A.

Another 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 and 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.

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 in a specific scenario, 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 derivative SGC [Wu *et al.*, 2019] eliminates the nonlinearities and collapses 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.

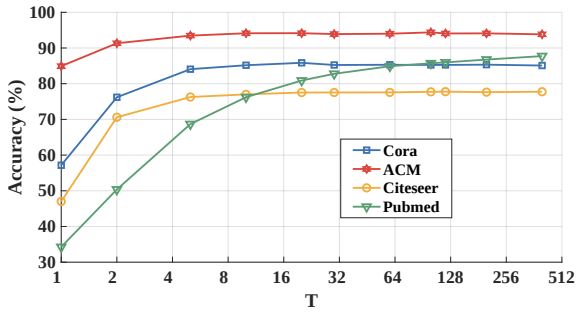


Figure 3: Impact of T

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 [3] 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 Notation, algorithm and source code

We list the frequently used notation in Table 8. 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 Discussion of Node Classification Experiments

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

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

Table 8: Frequently used notations in this paper

Notations	Descriptions
$\mathcal{G}$	Graph structure data
v_i	Single node in the graph
$\mathcal{V}$	Node set in the graph
N, C, d	Number of nodes, class number and feature dimensions
a_{ij}	Edge weight between nodes v_i and v_j, scalar
$\boldsymbol{A}$	Adjacent matrix of the graph, $\mathbb{R}^{N\times N}$
x_i	Feature vector of i-th node, $\mathbb{R}^{1\times d}$
$\boldsymbol{X}$	Entire attribute matrix in the graph, $\mathbb{R}^{N\times d}$
Y	One-hot labels for each node, $\mathbb{R}^{N\times C}$
d_i	Degree of a single node,scalar
$\boldsymbol{D}$	Diagonal matrix of the degree of each node, $\mathbb{R}^{N\times N}$
h_i	New feature of i-th node after convolution, $\mathbb{R}^{1\times d}$
$\boldsymbol{H}$	Entire attribute matrix after convolution, $\mathbb{R}^{N\times d}$
T	Time step in the clock-driven SNNs
O^{pre}	Spike of one node generated by encoder , $\mathbb{R}^{1\times d}$
O^{post}	Spike of one node generated by decoder, $\mathbb{R}^{1\times C}$
λ_i	j-th feature value of a single node, scalar
o_j	Basic spike unit generated by j-th feature value, equal to 0 or 1
V_m^t	Membrane potential at t-th time step in the decoder, $\mathbb{R}^{1\times C}$
ψ	Trainable weight matrix, $\mathbb{R}^{d\times C}$
τ_m	Time constant, hyperparameter, scalar
V_{reset}	Signed reset voltage, hyperparameter, scalar
V_{th}	Spiking threshold, hyperparameter, scalar

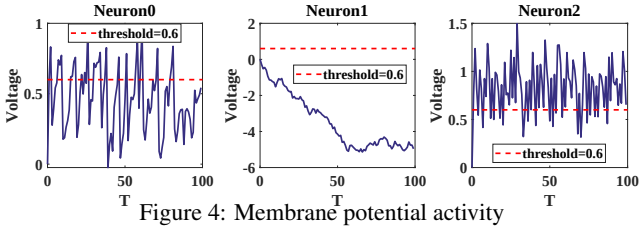


Figure 4: Membrane potential activity

mechanism [Veličković *et al.*, 2017]. Note 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.

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 citeseer 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.

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

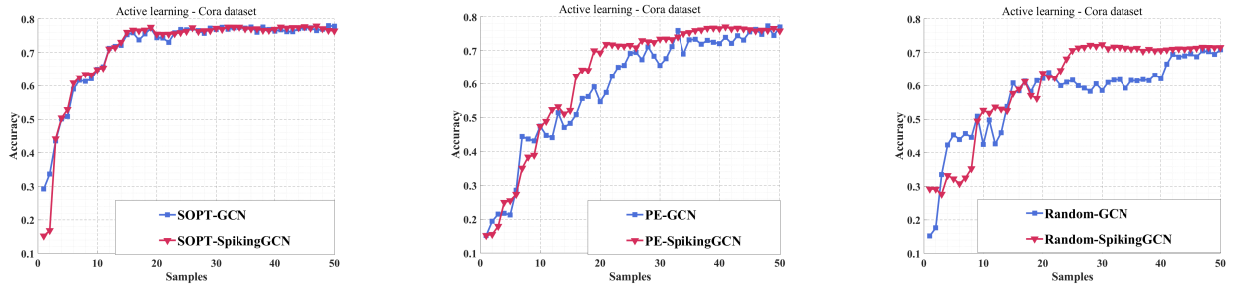
Table 9: The Area under the Learning Curve (ALC) on Cora and ACM datasets.

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

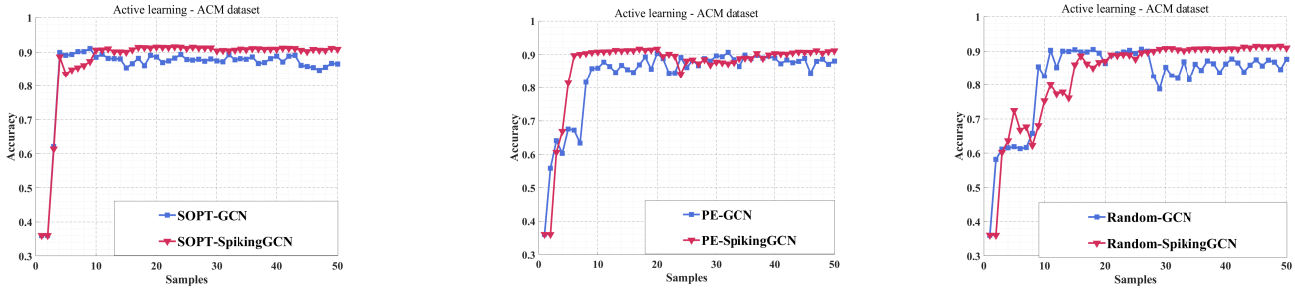
C.2 SpikingGCN for Active Learning

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



(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.

are selected. Results are reported on both Cora and ACM datasets.

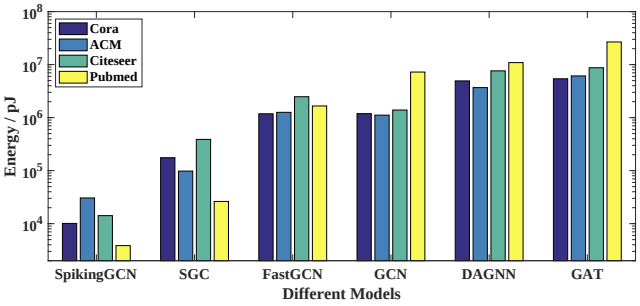


Figure 6: The energy consumption of SpikingGCN and baselines on their respective hardware.

The Area under the Learning Curve (ALC) [4] results are shown in Table 9. We provide the active learning curves of SpikingGCN and GCN in Fig. 5, which are consistent with the statistics reported in Table 9. 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.

C.3 Energy Efficiency Experiments

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

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

C.4 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 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:

$$V_m^t = V_m^{t-1}(1 - |\mathrm{H}\left(V_m^{t-1}\right)|) + X(t), \tag{15}$$

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.

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

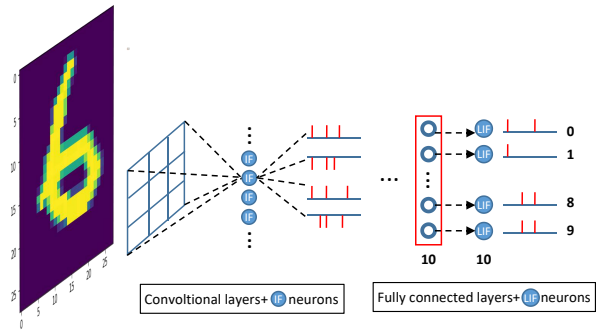


Figure 7: An extended model for deep SNNs

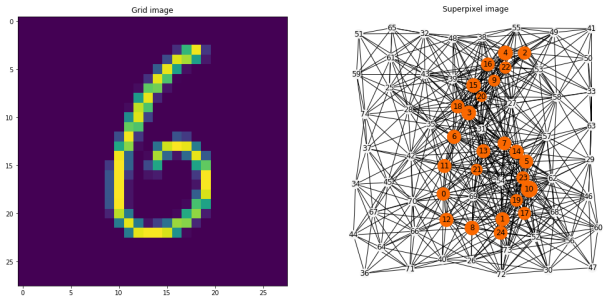
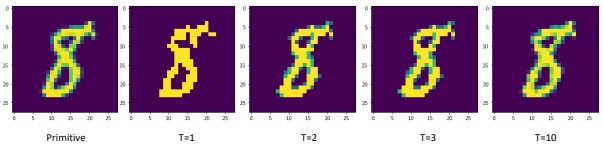


Figure 8: Comparison between grid images and superpixel images

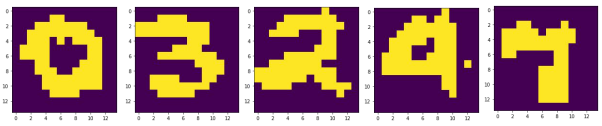
consists of connected nodes. Hence the classification task is defined as the prediction on the subgraphs. Another important distinction is that the superpixel images require to construct the connectivity between chosen nodes. A comparison between the grid and superpixel images is shown in Fig. 8, where 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.

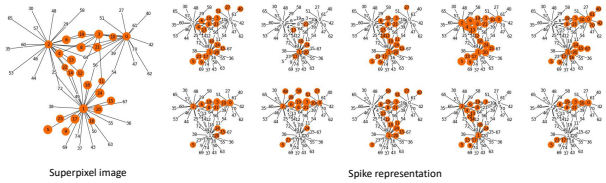
In addition, we also provide a unique perspective to understand the mechanism of our model. In particular, our spike encoder can be regarded as a sampling process using spike train representation. The scenario of the image graph provides us an ideal chance to visualize the data processing in our model. 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.

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.

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

[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.

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

[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.

[He *et al.*, 2020] 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*, pages 639–648. ACM, 2020.

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

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

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

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

[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.

[Lee *et al.*, 2018] 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.

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

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

[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.

[Rueckauer *et al.*, 2017] 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.

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

[Tavanaei *et al.*, 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.

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

[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.

[Xie *et al.*, 2020] 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.

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