Enhancing Graph Transformers with SNNs and Mutual Information

Ziyu Wang

The University of Tokyo, Japan
sltdwzy@outlook.com

Abstract. Although the integration of Graph Neural Networks (GNNs) and Transformers has demonstrated promising performance across various graph tasks, it remains computationally expensive. In contrast, brain-inspired Spiking Neural Networks (SNNs) offer an energy-efficient architecture due to their unique spike-based, event-driven paradigm. To address the high computational cost issue of Graph Transformers while maintaining the effectiveness to the maximum, in this paper, we propose a novel framework CSSGT, which leverages both the strength of Transformers and the computational efficiency of SNNs for graph tasks, trained under the graph contrastive learning framework. CSSGT comprises two key components: Mutual Information -based Graph Split (MIGS) and Spike-Driven Graph Attention (SDGA). MIGS is designed for sequential input of SNNs, splitting the graph while maximizing mutual information and minimizing redundancy. SDGA, tailored for graph data, exploits sparse graph convolution and addition operations, achieving low computational energy consumption. Extensive experiments on diverse datasets demonstrate that CSSGT converges within two epochs and outperforms various state-of-the-art models while maintaining low computational cost.

Keywords: Graph Neural Networks, Spiking Neural Networks, Transformers, Mutual Information, Graph Contrastive Learning.

1 Introduction

The integration of Graph Neural Networks (GNNs) and Transformers have achieved remarkable success across various graph tasks (19; 20; 6; 3; 33; 24). Additionally, this combination demonstrates high biological plausibility. GNNs are closely aligned with biological systems, while Transformers parallel current hippocampus models and recapitulate spatial representations found in the brain (23). However, despite their success, the combination of GNNs and Transformers remains computationally expensive, especially when dealing with large-scale datasets (28).

Brain-inspired spiking neural networks (SNNs), considered the third generation of neural networks (11), possess superior energy efficiency and biological plausibility due to the unique event-driven processing paradigm and the binary

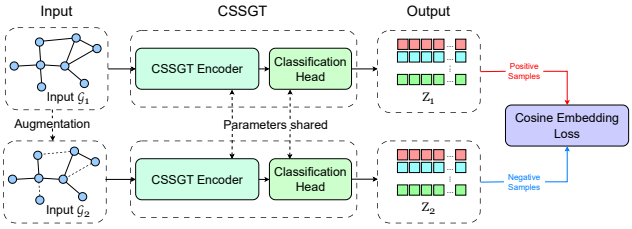


Fig. 1: Overview of CSSGT, trained under the graph contrastive learning paradigm. The detailed architecture is shown in Figure 2.

nature of spikes (15). In SNNs, neurons generate sparse and events-driven binary spikes (0 or 1) to communicate. Thus, during inference, the sparse and event-driven nature of spikes in SNNs eliminates the need for multiplication and significantly reduces computational cost, showing the outstanding energy-saving and memory-saving advantages when deployed on neuromorphic hardware (12; 4; 8). Moreover, SNNs exhibit memory capabilities due to their intrinsic temporal dynamics and the requirement for sequential input (14; 17). Given these characteristics, it is attractive to further incorporate SNNs to improve efficiency while developing highly biologically plausible models.

There has been research on incorporating SNNs into Transformers. Most Transformer-based SNNs prioritize maximizing performance instead of fully utilizing the energy efficiency of SNNs, often relying on Multiply-Accumulate (MAc) operations introduced by the vanilla Transformer architecture (5; 37; 36). (32) and (31) use the spike-driven paradigm, which only involves sparse Accumulate (Ac) operations, thus utilizing the advantage of SNNs to further reduce the computational cost. However, none of these works have incorporated GNNs or involved graph tasks. Efforts have also been made to combine SNNs and GNNs. The main challenge is how to fully exploit the benefits of SNNs for processing sequential input. (39) employs a strategy in which the graph is repeated multiple times to sequence the input, resulting in a large amount of redundant information without informative content, thereby limiting the performance of SNNs. Despite these efforts, the advantages of SNNs have yet to be fully exploited.

Our Contributions. In this work, We propose Contrastive learning -based Split Spiking Graph Transformer (CSSGT). We develop a novel mutual information -based graph split method (MIGS), which maximizes mutual information with class labels while minimizing redundancy between subsets. We theoretically proved that MIGS is guaranteed to fa within the range of optimal methods. Inspired by (32), we propose spike-driven graph attention (SDGA) specifically designed for graph data, which improves performance while maintaining low computational cost by adopting the spike-driven paradigm. To advance our work, we train CSSGT under the graph contrastive learning framework, further enhancing its biological plausibility and generalization ability to various datasets.

We conduct comprehensive evaluations on diverse node classification tasks, ranging from citation networks to Wikipedia networks. Experimental results

indicate that CSSGT converges within just 2 epochs, achieving performance surpassing various state-of-the-art (SOTA) models from different perspectives, while maintaining low energy consumption across various datasets. The main contributions of this work can be summarized as follows:

- We propose MIGS and SDGA tailored for SNNs in graph tasks. SDGA achieves significantly lower energy consumption compared to vanilla self-attention (19).
- Based on MIGS and SDGA, we develop CSSGT. CSSGT is trained under the graph contrastive learning framework, further enhancing its biological plausibility and generalizability.
- We conduct comprehensive evaluations on diverse datasets. The results demonstrate that CSSGT converges within two epochs and outperforms various SOTA models while maintaining low energy consumption.

2 Related Work

Spiking Neural Networks. SNNs benefit from the integration of deep learning and neuroscience. Many key SNN designs are inspired by various biological mechanisms (11; 15; 16; 26). However, the traditional backpropagation training algorithm cannot be applied directly to the discrete spikes, an alternative we adopt is surrogate gradient learning (9; 25). It avoids the non-differentiability of spike signals and approximates the backward gradients of the hard threshold function using a smooth activation function during backpropagation.

Efficient Transformers. Deploying Transformers with limited resources remains challenging due to their high computational cost (18). Typical optimizations include token (34), attention (2), and multi-head (13) approaches. Due to the attention mechanism's high computation scale, its optimization is a key focus. Removing softmax and modifying the operation and value of query, key, and value components are common methods for the optimization. Spiking Transformers typically follow this direction, incorporating SNNs and various computational orders into Query, Key, and Value components. However, most Spiking Transformers adopt MAc operations, limiting their efficiency benefits (37; 5; 36). (32) and (31) adopt the spike-driven paradigm, which involves only sparse Ac operations to further reduce the computational cost.

3 Method

In this section, we present CSSGT. We first briefly introduce the notation, mutual information, and spiking neuron layer, then present the overall architecture and the details of each component of CSSGT.

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ denote a graph with n nodes and f features, where $\mathcal{V} = \{v_i\}_{i=1}^{n}$, $\mathcal{E}$ and $\boldsymbol{X} \in \mathbb{R}^{n \times f}$ represent the set of nodes, the set of edges, and node feature matrix, respectively. Let $\boldsymbol{X}^{(i)}$ be the i-th partition of $\boldsymbol{X}$ after applying MIGS. Let $\boldsymbol{H}^{(i)}$, $\boldsymbol{Z}$, and $\boldsymbol{Y}$ denote the node feature matrix in

the hidden layers of the input partition $\boldsymbol{X}^{(i)}$, the final output of CSSGT, and the class labels, respectively.

Mutual Information. Mutual information is a measure of shared information between variables. Given two variables A and B, mutual information $\boldsymbol{I}(A;B)$ can be understood as the reduction in uncertainty of B with the knowledge of A. Formally, $\boldsymbol{I}(A;B)$ is defined as: $\boldsymbol{I}(A;B) = \mathrm{H}(A) - \mathrm{H}(A|B) = \mathrm{H}(B) - \mathrm{H}(B|A)$ where $\mathrm{H}(A), \mathrm{H}(B)$ are marginal entropies, and $\mathrm{H}(A|B), \mathrm{H}(B|A)$ are conditional entropies.

Spiking Neuron Layer. As a fundamental component of SNNs, the spiking neuron layer (SN) receives input signals, accumulates membrane potential and compares it with the threshold to determine whether to fire a spike. By default, we adopt the Parameterized Leaky Integrate-and-Fire (PLIF) neuron model in our work, whose dynamics can be described as:

$$\begin{aligned} U[t] &= H[t-1] + X[t], \\ S[t] &= \mathrm{Hea}(U[t] - u_{\mathrm{th}}), \\ H[t] &= V_{\mathrm{reset}} \cdot S[t] + \beta U[t](1 - S[t]), \\ \beta &= \exp(-\Delta t/\tau) \end{aligned}$$

where t denotes the time step of the input sequence, and $U[t]$ denotes the membrane potential produced by combining the spatial input $X[t]$ and the temporal input $H[t-1]$. PLIF model extends the LIF model by making key neuronal parameters learnable during training. Specifically, membrane constant τ, threshold u_{th}, and reset potential V_{reset} are all learnable parameters. When the membrane potential exceeds the threshold u_{th}, SN fires a spike (denoted by $S[t] = 1$); otherwise it does not (denoted by $S[t] = 0$). We use the Heaviside step function $\mathrm{Hea}(\cdot)$ to describe this mechanism, $\mathrm{Hea}(x) = 1$ when $x \geqslant 0$, and $\mathrm{Hea}(x) = 0$ otherwise. $H[t]$, V_{reset}, and $\beta < 1$ denote the temporal output, the reset potential, and the decay factor, respectively.

3.1 Overall Architecture

The architecture of CSSGT is shown in Figure 1 and Figure 2. The implementation of each component is detailed in Appendix C. The input is first split into T groups $\{\mathcal{G}_i\}_{i=1}^{T}$ using MIGS. Each group $\mathcal{G}_i$ then passes through a graph convolution block, Spiking Graph Position Encoding, and SDGA in order. Residual connections are incorporated in SGPE and SDGA. The outputs $\{\boldsymbol{H}_i\}_{i=1}^{T}$, are concatenated into $\boldsymbol{H}$, which is fed into a classification head and trained under the standard graph contrastive learning paradigm. The model can be written as

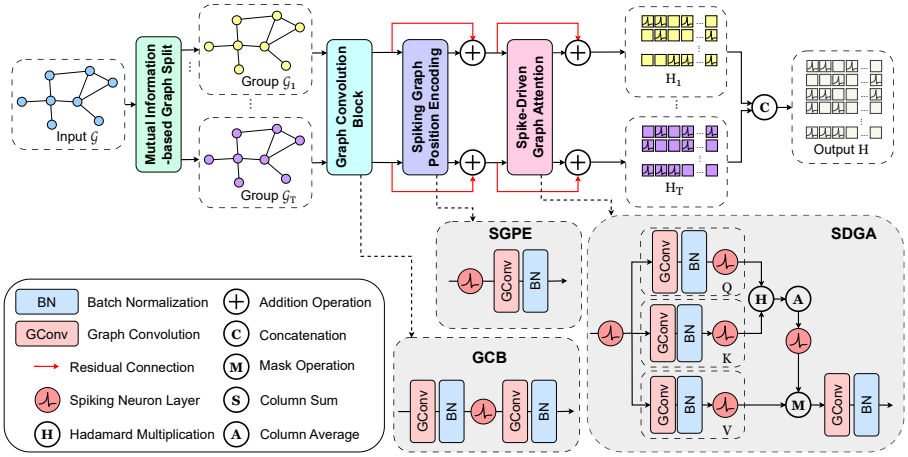


Fig. 2: The detailed architecture of CSSGT encoder.

follows:

$$
\begin{aligned}
\{\boldsymbol{X}^{(i)}\}_{i=1}^{T} &= \mathrm{MIGS}(\boldsymbol{X}), & \boldsymbol{X}^{(i)} &\in \mathbb{R}^{n\times f_i},\ \boldsymbol{X} \in \mathbb{R}^{n\times f} \\
\boldsymbol{H}_{\mathrm{GCB}}^{(i)} &= \mathrm{GCB}(\boldsymbol{X}^{(i)}), & \boldsymbol{H}_{\mathrm{GCB}}^{(i)} &\in \mathbb{R}^{n\times h} \\
\boldsymbol{H}_{\mathrm{SGPE}}^{(i)} &= \boldsymbol{H}_{\mathrm{GCB}}^{(i)} + \mathrm{SGPE}, & \boldsymbol{H}_{\mathrm{SGPE}}^{(i)} &\in \mathbb{R}^{n\times h} \\
\boldsymbol{H}^{(i)} &= \boldsymbol{H}_{\mathrm{SGPE}}^{(i)} + \mathrm{SDGA}, & \boldsymbol{H}_{\mathrm{Att}}^{(i)} &\in \mathbb{R}^{n\times h} \\
\boldsymbol{H} &= \mathrm{Concat}(\{\boldsymbol{H}^{(i)}\}_{i=1}^{T}), & \boldsymbol{H} &\in \mathbb{R}^{T\times n\times h} \\
\boldsymbol{Z} &= \mathrm{CH}(\boldsymbol{H}), & \boldsymbol{Z} &\in \mathbb{R}^{T\times n\times c}
\end{aligned}
$$

Where h is the hidden dimension, T is the number of partitions, n is the number of nodes and f_i is the feature dimension of the i-th partition. CH is classification head, which is implemented using a linear layer.

3.2 Mutual Information-based Graph Split

We address the challenge of processing graph data to meet SN's requirement for sequential input. A common approach is to split the graph into multiple groups along the feature dimension, ensuring a uniform distribution of nodes across these groups. This reduces inter-group variability, simplifies further processing, and lowers computational complexity. The key challenge, as mentioned earlier, is organizing the feature distribution to allow SNs to effectively integrate information over time while minimizing information loss.

We propose MIGS, a graph splitting method based on information theory. Along the feature dimension, we split $\boldsymbol{X}$ into T disjoint subsets $\{\boldsymbol{X}^{(i)}\}_{i=1}^{T}$, such that each subset $\boldsymbol{X}^{(t)} \in \mathbb{R}^{N\times F_t}$ maximizes the mutual information $\boldsymbol{I}(\boldsymbol{X}^{(t)}; \boldsymbol{Y})$

Algorithm 1 Mutual Information-based Graph Split

Input:
Feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times f}$
Class labels $\boldsymbol{Y} \in \{1, 2, \cdots, C\}^N$
Number of partitions T
Regularization parameter λ
Output:
Partitions $\{\boldsymbol{X}^{(i)}\}_{i=1}^{T}$
Initialize:
Mutual information $\boldsymbol{I}(x_i; \boldsymbol{Y})$, $\boldsymbol{I}(x_i; x_{i'})$
Empty partitions $\{\boldsymbol{X}^{(i)}\}_{i=1}^{T}$
Unassigned features $U = \{x_i\}_{i=1}^{f}$
for $t = i$ to T **do**
  Select x_f with highest $I(x_f; \boldsymbol{Y})$
  Assign x_f to partition $\boldsymbol{X}^{(i)}$
  Remove x_f from U
end for
while U is not empty **do**
  for each unassigned feature $x_f \in U$ **do**
    for each partition $i = 1$ to T **do**
      Calculate mutual information gain
      $\Delta I^{(i)}_{\text{label}}(x_f) = I(x_f; \boldsymbol{Y})$
      Calculate redundancy
      $\Delta I^{(i)}_{\text{rdd}}(x_f) = \sum_{x_{f'} \in \boldsymbol{X}^{(t)}} I(x_f; x_{f'})$
      Compute net gain $\Delta I^{(i)}(x_f)$
      $= \Delta I^{(i)}_{\text{label}}(x_f) - \lambda \Delta I^{(i)}_{\text{rdd}}(x_f)$
    end for
    Assign x_f to partition
    $t^* = argmax_i \Delta I^{(i)}(x_f)$
    Remove x_f from U
  end for
end while

with the class labels $\boldsymbol{Y}$, ensuring that the most discriminative features are preserved and propagated through the network. The partition also ensures each subset minimizes the redundancy between subsets, reducing information overlap.

We start by formulating the optimization problem (proved in Appendix B.1):

Proposition 1. *Maximizing the mutual information $\boldsymbol{I}(\boldsymbol{Z}; \boldsymbol{Y})$ between $\boldsymbol{Z}$ and $\boldsymbol{Y}$ is equivalent to solving the following optimization problem, where λ is a regularization parameter:*

$$\boldsymbol{X}^{(t)} = argmax \sum_{t=1}^{T} \boldsymbol{I}(\boldsymbol{X}^{(t)}; \boldsymbol{Y}) - \lambda \sum_{t \neq s} \boldsymbol{I}(\boldsymbol{X}^{(t)}; \boldsymbol{X}^{(s)}), \forall t, s \in \{1, 2, \cdots, T\}$$

We employ a greedy feature partitioning algorithm to solve this optimization problem. The algorithm begins by computing the mutual information between each feature and the class labels $\boldsymbol{I}(x_f; \boldsymbol{Y}) = \mathrm{H}(x_f) - \mathrm{H}(x_f|\boldsymbol{Y})$, as well as the mutual information between all pairs of features $\boldsymbol{I}(x_f; x_{f'}) = \mathrm{H}(x_f) - \mathrm{H}(x_f|x_{f'})$. Each partition $\boldsymbol{X}^{(i)}$ is initialized with the feature that has the highest $\boldsymbol{I}(x_f; \boldsymbol{Y})$.

At each iteration, we assign a feature to the partition that yields the maximum net gain $\Delta \boldsymbol{I}^{(t)}(x_f) = \Delta \boldsymbol{I}^{(t)}_{\text{label}}(x_f) - \lambda \Delta \boldsymbol{I}^{(t)}_{\text{rdd}}$, where $\Delta \boldsymbol{I}^{(t)}_{\text{label}}(x_f) = \boldsymbol{I}(x_f; \boldsymbol{Y})$, $\Delta \boldsymbol{I}^{(t)}_{\text{rdd}}(x_f) = \sum_{x_{f'} \in \boldsymbol{X}^{(t)}} \boldsymbol{I}(x_f; x_{f'})$. By selecting the partition that maximizes the net gain for each feature, the algorithm ensures that each partition contains highly informative and minimally redundant features. This method effectively constructs feature subsets that, when processed through SNs, minimize information loss and enhance performance by providing diverse and complementary

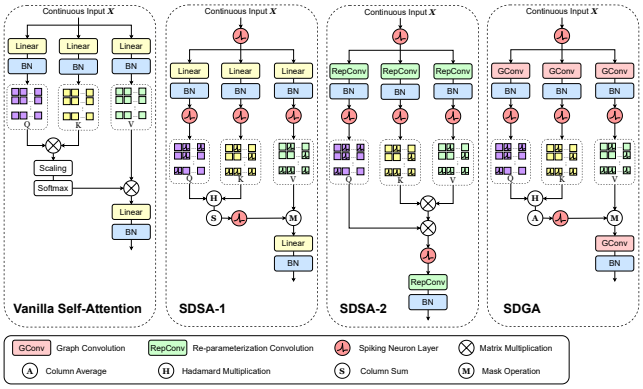


Fig. 3: Comparison between Vanilla Self-Attention (VSA) (19), Spike-Driven Self-Attention-1 (SDSA-1) (32), Spike-Driven Self-Attention-2 (SDSA-2) (31), and our Spike-Driven Graph Attention (SDGA). The key differences are: **(a)** we use graph convolution layer instead of re-parameterization convolution layer and linear layer to process graph data; **(b)** we use column-wise average operation instead of column-wise sum operation.

information across partitions. A comprehensive description is provided in Algorithm 1.

We further conclude that the greedy algorithm is guaranteed to fall within the range of optimal solutions, as shown in Proposition 2 (proved in Appendix B.2).

Proposition 2. *The greedy algorithm given above approaches the optimal solution. Specifically, For the maximization optimization of the objective function f in Proposition 1, the greedy algorithm partition solution S_{Greedy} and the optimal solution S^* satisfy:*

$$f(S_{Greedy}) \geqslant (1 - \frac{1}{e}) f(S^*)$$

3.3 Spike-Driven Graph Attention

In this section, we present the architecture of SDGA. The detailed energy analyses are provided in Section 4.5 and Appendix A.

Given the input $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, to leverage the computational efficiency, we first apply SN to the input:

$$\boldsymbol{X}_s = \mathrm{SN}(\boldsymbol{X})$$

Instead of using the convolution layer employed by Spike-Driven Self-Attentions (32; 31), or the linear layer adopted by the Vanilla Self-Attention, SDGA employs

graph convolution to obtain the query $\boldsymbol{Q}$, key $\boldsymbol{K}$ and value $\boldsymbol{V}$:

$$\begin{aligned}\boldsymbol{Q} &= \mathrm{BN}(\mathrm{GConv}_{\boldsymbol{Q}}(\boldsymbol{X}_s)),\\ \boldsymbol{K} &= \mathrm{BN}(\mathrm{GConv}_{\boldsymbol{K}}(\boldsymbol{X}_s)),\\ \boldsymbol{V} &= \mathrm{BN}(\mathrm{GConv}_{\boldsymbol{V}}(\boldsymbol{X}_s))\end{aligned}$$

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n\times h}$, $\mathrm{GConv}_{\boldsymbol{Q}}$, $\mathrm{GConv}_{\boldsymbol{K}}$, $\mathrm{GConv}_{\boldsymbol{V}}$ are graph convolution layers, BN is batch normalization layer, h is the hidden dimension. Since $\boldsymbol{X}_s$ is a sparse binarized matrix, the computation cost of graph convolution remains low. Then we apply SN again:

$$\boldsymbol{Q}_s = \mathrm{SN}(\boldsymbol{Q}),\ \boldsymbol{K}_s = \mathrm{SN}(\boldsymbol{K}),\ \boldsymbol{V}_s = \mathrm{SN}(\boldsymbol{V})$$

Inspired by SDSA-1 (32), we compute the Hadamard product between $\boldsymbol{Q}_s$ and $\boldsymbol{K}_s$. This operation can be regarded as energy-free, as the Hadamard product between spikes is equivalent to the element-wise mask operation:

$$\boldsymbol{\alpha} = \boldsymbol{Q}_s \odot \boldsymbol{K}_s,\ \boldsymbol{\alpha} \in \mathbb{R}^{n\times h}$$

Given that the threshold of SN is in the range $[0, 1]$, instead of column-wise sum operation, we use column-wise average operation before SN, ensuring the input is scaled appropriately within the range $[0, 1]$. We then compute the mask operation with $\boldsymbol{V}_s$:

$$\overline{\boldsymbol{\alpha}} = \mathrm{SN}(\mathrm{Average}(\boldsymbol{\alpha})),\ \overline{\boldsymbol{\alpha}} \in \mathbb{R}^{n\times 1}$$

$$\boldsymbol{V}_{\mathrm{mask}} = \mathrm{Mask}(\overline{\boldsymbol{\alpha}},\ \boldsymbol{V}_s),\ \boldsymbol{V}_{\mathrm{mask}} \in \mathbb{R}^{n\times h}$$

Finally we apply graph convolution to $\boldsymbol{V}_{\mathrm{mask}}$ to obtain the SDGA:

$$\mathrm{SDGA} = \mathrm{BN}(\mathrm{GConv}(\boldsymbol{V}_{\mathrm{mask}}))$$

We outline the architecture of SDGA and compare it with other attention mechanisms (see Figure 3). The intuition behind SDGA is that graph convolution is better suited for graph data, with energy consumption comparable to MLP, significantly lower than traditional convolution. Therefore, we integrate graph convolution into the attention mechanism, and the column-wise average operation ensures the input is appropriately scaled within the range $[0, 1]$.

It is important to note that the single-head SDGA can be easily extended to multi-head SDGA (MSDGA). After the first graph convolution, we split the query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$ into H multiple heads: $\boldsymbol{Q} = (\hat{\boldsymbol{Q}}_1, \hat{\boldsymbol{Q}}_2, \cdots, \hat{\boldsymbol{Q}}_H)$, $\boldsymbol{K} = (\hat{\boldsymbol{K}}_1, \hat{\boldsymbol{K}}_2, \cdots, \hat{\boldsymbol{K}}_H)$, $\boldsymbol{V} = (\hat{\boldsymbol{V}}_1, \hat{\boldsymbol{V}}_2, \cdots, \hat{\boldsymbol{V}}_H)$ Where $\hat{\boldsymbol{Q}}_i, \hat{\boldsymbol{K}}_i, \hat{\boldsymbol{V}}_i \in \mathbb{R}^{n\times h/H}$. Then we apply SDGA on each head and concatenate the outputs to obtain the final output MSDGA:

$$\mathrm{MSDGA} = (\mathrm{SDGA}(\hat{\boldsymbol{Q}}_1, \hat{\boldsymbol{K}}_1, \hat{\boldsymbol{V}}_1), \cdots, \mathrm{SDGA}(\hat{\boldsymbol{Q}}_H, \hat{\boldsymbol{K}}_H, \hat{\boldsymbol{V}}_H))$$

3.4 Graph Convolution Block

The graph convolution block (GCB) contains two graph convolution layers (GConv) and two batch normalization layers (BN):

$$\hat{\boldsymbol{H}}_{\mathrm{GCB}}^{(i)} = \mathrm{BN}_1(\mathrm{GConv}_1^{(i)}(\boldsymbol{X}^{(i)})),\ i \in \{1, \cdots, T\}$$

$$\boldsymbol{H}_{\mathrm{GCB}}^{(i)} = \mathrm{BN}_2(\mathrm{GConv}_2(\mathrm{SN}(\hat{\boldsymbol{H}}_{\mathrm{GCB}}^{(i)})))$$

Different graph subsets are processed sequentially by GCB. For each input subset, the first graph convolution layer $\mathrm{Conv}_1^{(i)}$ has different inputs but identical output dimensionality. The second graph convolution layer Conv_2 has the same input and output dimensionality with parameters shared for different input subsets.

3.5 Spiking Graph Position Encoding

In spiking graph position encoding, similar to SDGA, we first apply the spiking neuron layer to reduce computational cost, followed by graph convolution to encode the position information. Specifically, for the input $\boldsymbol{X}$, spiking graph position encoding (SGPE) can be expressed as:

$$\mathrm{SGPE} = \mathrm{BN}(\mathrm{GConv}(\mathrm{SN}(\boldsymbol{X})))$$

3.6 Graph Contrastive Learning

We follow the conventional graph contrastive learning framework for training. To generate two different views, we apply edge dropout and cosine loss (1) on the final outputs:

$$\mathrm{loss}(x, y) = \begin{cases} 1 - \cos(\boldsymbol{Z}_1, \boldsymbol{Z}_2), & \text{if } y = 1 \\ \max(0, \cos(\boldsymbol{Z}_1, \boldsymbol{Z}_2) - \mathrm{margin}), & \text{if } y = -1 \end{cases}$$

Where $\boldsymbol{Z}_1,\ \boldsymbol{Z}_2$ are the final output pairs, $x = (\boldsymbol{Z}_1, \boldsymbol{Z}_2)$ represents a pair of graph representations, and y determines whether the pairs are positive or negative. Cosine loss maximizes the similarity between positive pairs and minimizes it for negative pairs by computing the cosine similarity.

4 Experiments

In this section, we first present the experimental settings(Section 4.1). We then evaluate the test accuracy and convergence performance of CSSGT on node classification tasks across diverse datasets, comparing it with various models (Sections 4.2 and 4.3). We also conduct ablation studies on SDGA, MIGS, and the impact of hyperparameters, including spiking threshold, the number of GNN layers, and the type of SNs (Section 4.4). Finally, we provide the analysis of energy efficiency across various attention mechanisms and models (Section 4.5). Implementation details and additional experiment results are provided in Appendix C and Appendix E.

4.1 Setup

Datasets. The experiments are conducted on six commonly used graph datasets: *Cora*, *Citeseer*, *Pubmed*, *Chameleon*, *Squirrel*, and *obgn-Arxiv*. The first three are medium-sized citation networks with high homophily ratios. *Chameleon* and *Squirrel* are Wikipedia networks and identified as heterophilic graphs. *obgn-Arxiv* is a well-known large-scale graph. The details are provided in Appendix D.

Baselines. We compare CSSGT with a wide range of SOTA baselines from various perspectives. In terms of classical supervised GNNs, We compare against GCN (7), GAT (21), and an advanced GNN, JKNet-GAT (29), in which GAT is used as the backbone of JKNet. In terms of unsupervised models, we compare with SOTAs including DGI (22), GRACE (38), and CCA-SSG (35). In terms of spiking-based models, we compare against GC-SNN (30), GA-SNN (30), and SpikingNet (10). In terms of Transformer-based models, we compare with various SOTAs including NodeFormer (24), Graphormer (33), GraphTrans (27), and SGFormer. Since the original Graphormer and GraphTrans are too large to scale on all datasets, we compare with the small-sized versions, $\text{Graphormer}_{\text{small}}$ (3 layers and 8 heads), $\text{GraphTrans}_{\text{small}}$ (3 layers and 4 heads), $\text{Graphormer}_{\text{ultrasmall}}$ (2 layers and 1 head), $\text{GraphTrans}_{\text{ultrasmall}}$ (2 layers and 1 head).

4.2 Node Classification

Table 1 reports the results of all the models. Results demonstrate that CSSGT effectively handles both homophilic and heterophilic graphs, consistently outperforming all competitors across every dataset. Notably, CSSGT achieves its highest improvements on *Citeseer* over Transformer-based models, and on *Pubmed* over classical and unsupervised models. Since both of these datasets have high feature dimensions, a possible explanation for the significant improvement of CSSGT is that MIGS splits the high-dimensional input into multiple lower-dimensional inputs, which helps to reduce information loss from the input data.

4.3 Convergence Analysis

We further study the convergence performance of CSSGT in node classification task. To ensure the clarity of the figures, we report the accuracy curves on *Cora*, *Pubmed*, *Chameleon*, and *Squirrel*, comparing CSSGT with NodeFormer, GAT, JKNet-GAT, and GCN. The results are presented in Figure 4. Results demonstrate that, across all the datasets, CSSGT consistently converges within two epochs and achieves the highest accuracy. In contrast, on medium-sized datasets *Cora* and *Citeseer*, competitors require at least 30 epochs to converge. On larger datasets *Chameleon* and *Squirrel*, the performance of competitors barely improves after 30 epochs. A possible explanation is that the combination of MIGS and SDGA effectively leverages the advantages of SNs for processing sequential data, as the input data in each epoch is split into multiple partitions, serving as sequential input to the SNs.

Methods	U	S	T	Cora	Citeseer	Pubmed	Chameleon	Squirrel	obgn-Arxiv
GCN				81.5±0.1	71.3±0.2	79.0±0.2	54.3±3.0	38.6±1.8	71.8±0.3
GAT				83.0±0.7	72.5±0.7	79.0±0.3	51.2±3.1	35.6±2.1	72.1±0.1
JKNet-GAT				83.3±0.4	72.7±0.3	79.9±0.5	51.5±1.3	40.1±1.5	72.3±0.2
DGI	✓			82.3±0.6	71.8±0.7	76.8±0.6	60.3±0.7	39.4±1.1	70.3±0.2
GRACE	✓			81.9±0.4	71.2±0.5	80.6±0.4	58.1±0.9	41.4±0.7	71.5±0.4
CCA-SSG	✓			84.0±0.4	73.1±0.3	81.0±0.4	57.4±1.4	42.2±1.0	71.2±0.5
GC-SNN		✓		80.7±0.6	69.9±0.9	OOM	53.1±1.4	39.4±1.1	66.4±0.3
GA-SNN		✓		79.9±0.6	69.1±0.5	OOM	54.6±0.9	38.7±1.3	66.7±0.2
SpikingNet		✓		82.6±0.5	71.4±0.5	78.6±0.3	55.7±1.9	40.7±1.8	67.5±1.8
Nodeformer			✓	83.2±0.9	72.5±1.1	79.9±1.0	49.6±4.1	38.5±1.5	59.9±0.4
$\text{Graphormer}_{\text{small}}$			✓	75.8±1.1	65.6±0.6	OOM	54.9±2.8	40.9±2.5	OOM
$\text{Graphormer}_{\text{ultrasmall}}$			✓	74.2±0.9	63.6±1.0	OOM	54.2±2.4	33.9±1.4	OOM
$\text{GraphTrans}_{\text{small}}$			✓	80.7±0.9	69.5±0.7	OOM	55.7±3.3	41.0±2.8	OOM
$\text{GraphTrans}_{\text{ultrasmall}}$			✓	81.7±0.6	70.2±0.8	77.4±0.5	55.2±2.9	40.6±2.4	OOM
SGFormer			✓	84.5±0.8	72.6±0.2	80.3±0.6	56.9±3.9	41.8±2.2	72.6±0.1
CSSGT (ours)	✓	✓	✓	**87.3±1.2**	**77.6±1.0**	**86.9±0.8**	**60.7±1.2**	**42.6±1.9**	**73.2±0.7**

Table 1: Node classification test accuracy. U, S, T, and OOM stand for unsupervised, spiking-based, Transformer-based, and out of memory, respectively.

4.4 Ablation Study

We proceed to conduct ablation studies on MIGS, SDGA, SGPE, and the impact of hyperparameters, including spiking threshold, number of GNN layers, and type of SNs. Due to space limitations, we present the figures of accuracy curves and the table of results on dataset *Chameleon*, *Squirrel*, *Cora*, and *Citeseer*. The implementation details and additional results are provided in Appendix C and Appendix E.

Mutual Information-based Graph Split. Figure 5a presents the accuracy curves of CSSGT with varying numbers of split groups, ranging from 1 to 100. The results consistently indicate that CSSGT is relatively insensitive to the number of groups when the number is greater than 20. However, when the number of groups is less than 20, both performance and convergence speed improve as the number of groups increases.

Model	Cora	Citeseer	Chameleon	Squirrel
w/o SDGA	85.4	73.8	60.5	40.0
w/o SGPE	85.4	75.4	59.6	38.2
1-head	87.2	**76.5**	61.2	**41.3**
2-head	87.5	76.3	61.2	40.6
4-head	87.3	**76.5**	**61.4**	**41.3**
8-head	**87.6**	**76.5**	61.2	40.8

Table 2: Ablation on SDGA and SGPE

Spike-Driven Graph Attention and Spiking Graph Position Encoding.

Table 2 presents the accuracy comparison of CSSGT with varying numbers of attention heads, from 1 to 8, without SDGA, and without SGPE. The results demonstrate that, both SDGA and SGPE consistently enhance the performance of CSSGT. The 4-head SDGA achieves the best performance, surpassing CSSGT

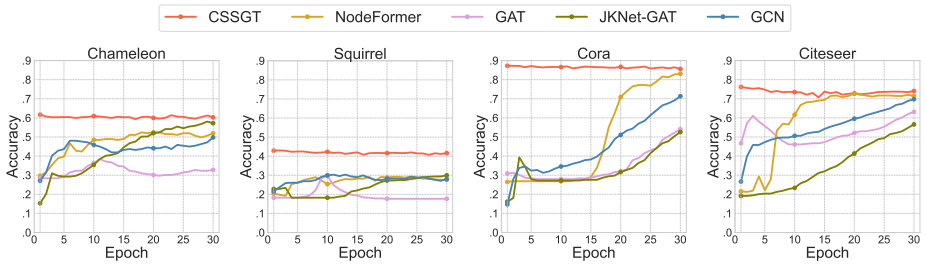
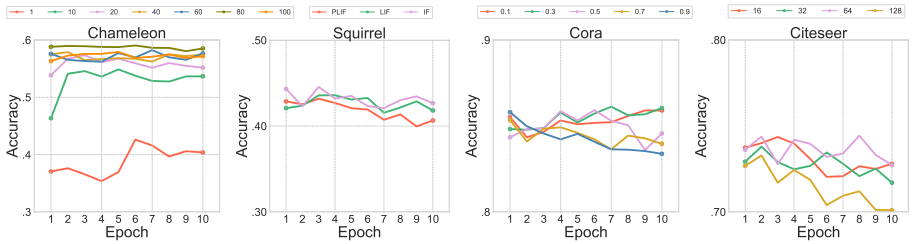


Fig. 4: Node classification accuracy curves comparison on *Chameleon*, *Squirrel*, *Cora*, and *Citeseer* datasets. CSSGT consistently converges within two epochs and achieves the highest accuracy.



(a) MIGS (b) Spiking type (c) Spiking threshold (d) Dimensions

Fig. 5: Ablation studies. **(a)** MIGS with different number of partitions; **(b)** type of the spiking model type in SNs; **(c)** threshold of SNs; **(d)** hidden dimension of all graph convolution layers.

without SDGA by 3.7% on *Citeseer* and CSSGT without SGPE by 3.3% on *Squirrel*.

Impact of spiking type and hyperparameters. Figure 5b, 5c, and 5d present the accuracy curves of CSSGT with different types of SNs: Parameterized Leaky-and-Integrate (PLIF) model, Leaky-and-Integrate (LIF) model, and Integrate-and-Fire (IF) model; different spiking thresholds: 0.1, 0.3, 0.5, 0.7, 0.9; and different hidden dimensions: 16, 32, 64, and 128, respectively. The results indicate that CSSGT is relatively insensitive to spiking type and hyperparameters. Nevertheless, a spiking threshold of 0.7, and hidden dimension of 64 outperforms other settings.

4.5 Efficiency Analysis

Attention analysis. We calculate the complexity, total number of operations, and energy consumption of SDGA compared with VSA, SDSA-1, and SDSA-2. We assume the input matrix is of size $n \times h$. The results are summarized in Table 3, and the detailed calculation process is provided in Appendix A. The results show that SDGA achieves the lowest number of operations and energy

Architecture	Complexity	Cora	Citeseer	Chameleon	Squirrel
VSA	$O(hn^2)$	994M/4.57mJ	1491M/6.86mJ	707M/3.25mJ	3608M/16.60mJ
SDSA-1	$O(h^2n)$	559M/0.50mJ	655M/0.59mJ	571M/0.51mJ	1202M/1.08mJ
SDSA-2	$O(h^2n)$	581M/0.52mJ	687M/0.62mJ	610M/0.55mJ	1265M/1.14mJ
SDGA (ours)	$O(h^2n)$	**198M/0.18mJ**	**232M/0.21mJ**	**203M/0.18mJ**	**426M/0.38mJ**

Table 3: Comparison of the number of operations and energy consumption between different attention mechanisms. The results are presented in the form of operations/energy.

Architecture	Cora	Citeseer	Chameleon	Squirrel
CCA-SSG	4.40B/20.08mJ	13.12B/60.24mJ	5.76B/26.48mJ	11.84B/54.64mJ
SGFormer	0.57B/2.62mJ	1.67B/7.66mJ	0.74B/3.39mJ	1.53B/7.03mJ
CSSGT (ours)	0.93B/**0.84mJ**	2.09B/**1.88mJ**	1.16B/**1.04mJ**	2.38B/**2.14mJ**

Table 4: Comparison of the number of operations and energy consumption between CCA-SSG, SGFormer, and CSSGT. The results are presented in the form of operations/energy.

consumption across all datasets, with up to 43.68× lower energy consumption compared to VSA, and around 3× lower energy consumption compared to SDSA-1 and SDSA-2.

Model analysis. We further compare the number of operations and energy consumption of CSSGT with CCA-SSG and SGFormer, two of the most efficient architectures in contrastive learning-based and Transformer-based models. The results are presented in Table 4, with detailed calculations provided in Appendix A. The results show that CSSGT is significantly more efficient compared to CCA-SSG. Although CSSGT has a larger number of operations compared to SGFormer, since the operations are Ac in CSSGT, while CCA-SSG and SGFormer rely on MAc, the energy consumption of CSSGT is even 3× lower.

5 Conclusion and Discussion

In this paper, to address the high computational cost issue of Graph Transformers during inference, we aim to achieve effective performance while fully leveraging the efficient nature of SNNs. Therefore, we first propose the novel MIGS minimizing the information loss to ensure effectiveness, supported by solid theoretical foundations. We then incorporate SNNs globally in Graph Transformers a spike-driven paradigm to ensure efficiency, which is also, to our best knowledge, the first study in this direction. This incorporation consists of SDGA, SGPE, and GCB. We train the network under GCL and name it CSSGT.

Discussion on performance. A possible explanation for CSSGT's great performance lies in the combination of SNNs and MIGS. In CSSGT, we intro-

duce an SN before each graph convolution operation, which sparsifies the data and limits the operations to simple Accumulate operation (detailed in Appendix A), significantly reducing energy consumption. While some information loss is inevitable, this approach introduces two key advantages beyond efficiency: a temporal dimension, along which MIGS encodes the information of the input; and the accumulation dynamics of SNs, which provide memory of previous inputs.

Discussion on experiments. Given the theory of GCL and the linear evaluation protocol used for evaluation, which is a simple linear classifier on top of the frozen representations for a specific downstream task, as detailed in the implementation details section, the model's representations will generalize to various downstream tasks, even with a minimal and task-agnostic evaluation setup. Thus, it is standard practice to use the node classification evaluation task as a proxy to assess the representation of the model. We note that CSSGT can potentially be extended to other graph tasks such as dynamic graph node classification and graph classification.

Discussion on GCL. The key insight of using Contrastive learning is that SNNs inherently introduce the temporal dimension into the graph data, resulting in additional information and relevance in the time domain. However, the binary nature of SNNs causes the loss of information for the supervised-based method, which makes it hard to handle the relevance of data. Contrastive learning provides a framework to explicitly leverage temporal correlations, fully aligning with the SNNs' strength in processing sequential data. Moreover, since the spikes generated by SNNs often exhibit high sparsity (please refer to Appendix E.6), contrastive learning can handle the sparse data well by enhancing the ability to utilize the relevant information. Lastly, Contrastive learning improves robustness, which can be crucial given SNNs' binary nature.

Limitation. Although the calculation of mutual information in MIGS is required only once for a given dataset, it remains relatively computationally expensive.

6 Acknowledgement

My heartfelt thanks go to my advisor, Dr. Watanabe, whose kindness has challenged me to grow. And I hope this paper marks a hopeful beginning to my journey of exploration. *The future, for which I really worked, is mine.*

This work has been supported by the Mohammed bin Salman Center for Future Science and Technology for Saudi-Japan Vision 2030 at The University of Tokyo (MbSC2030) and JSPS KAKENHI Grant Number 23K25257.

Bibliography

[1] Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: Proceedings of the 37th International Conference on Machine Learning (2020)

[2] Choromanski, K.M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J.Q., Mohiuddin, A., Kaiser, L., Belanger, D.B., Colwell, L.J., Weller, A.: Rethinking attention with performers. In: International Conference on Learning Representations (2021)

[3] Dwivedi, V.P., Bresson, X.: A generalization of transformer networks to graphs. AAAI Workshop on Deep Learning on Graphs: Methods and Applications (2021)

[4] Furber, S., Galluppi, F., Temple, S., Plana, L.: The spinnaker project. Proceedings of the IEEE **102**, 652–665 (2014)

[5] Han, M., Wang, Q., Zhang, T., Wang, Y., Zhang, D., Xu, B.: Complex dynamic neurons improved spiking transformer network for efficient automatic speech recognition (2023)

[6] Hu, Z., Dong, Y., Wang, K., Sun, Y.: Heterogeneous graph transformer. In: Proceedings of The Web Conference 2020 (2020)

[7] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2017)

[8] Krestinskaya, O., James, A.P., Chua, L.O.: Neuromemristive circuits for edge computing: A review. IEEE Transactions on Neural Networks and Learning Systems **31**, 4–23 (2020)

[9] Lee, J., Delbrück, T., Pfeiffer, M.: Training deep spiking neural networks using backpropagation. Frontiers in Neuroscience (2016)

[10] Li, J., Yu, Z., Zhu, Z., Chen, L., Yu, Q., Zheng, Z., Tian, S., Wu, R., Meng, C.: Scaling up dynamic graph representation learning via spiking neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence (2023)

[11] Maass, W.: Networks of spiking neurons: The third generation of neural network models. Neural Networks **10**, 1659–1671 (1997)

[12] Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., Jackson, B.L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S.K., Appuswamy, R., Taba, B., Amir, A., Flickner, M.D., Risk, W.P., Manohar, R., Modha, D.S.: A million spiking-neuron integrated circuit with a scalable communication network and interface. Science **345**, 668–673 (2014)

[13] Michel, P., Levy, O., Neubig, G.: Are sixteen heads really better than one? (2019)

[14] Ponulak, F., Kasiński, A.: Introduction to spiking neural networks: Information processing, learning and applications. Acta neurobiologiae experimentalis **71**, 409–33 (2011)

[15] Roy, K., Jaiswal, A.R., Panda, P.: Towards spike-based machine intelligence with neuromorphic computing. Nature **575**, 607 – 617 (2019)
[16] Schuman, C., Kulkarni, S., Parsa, M., Mitchell, J., Date, P., Kay, B.: Opportunities for neuromorphic computing algorithms and applications. Nature Computational Science **2**, 10–19 (2022)
[17] Tavanaei, A., Ghodrati, M., Kheradpisheh, S.R., Masquelier, T., Maida, A.: Deep learning in spiking neural networks. Neural Networks **111**, 47–63 (2019)
[18] Tay, Y., Dehghani, M., Bahri, D., Metzler, D.: Efficient transformers: A survey. ACM Comput. Surv. (2022)
[19] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems (2017)
[20] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. 6th International Conference on Learning Representations (2017)
[21] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018)
[22] Veličković, P., Fedus, W., Hamilton, W.L., Liò, P., Bengio, Y., Hjelm, R.D.: Deep graph infomax. In: International Conference on Learning Representations (2019)
[23] Whittington, J.C.R., Warren, J., Behrens, T.E.: Relating transformers to models and neural representations of the hippocampal formation. In: International Conference on Learning Representations (2022)
[24] Wu, Q., Zhao, W., Li, Z., Wipf, D., Yan, J.: Nodeformer: A scalable graph structure learning transformer for node classification. In: Advances in Neural Information Processing Systems (2022)
[25] Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., Shi, L.: Direct training for spiking neural networks: Faster, larger, better. Proceedings of the AAAI Conference on Artificial Intelligence (2019)
[26] Wu, Y., Zhao, R., Zhu, J., Chen, F., Xu, M., Li, G., Song, S., Deng, L., Wang, G., Zheng, H., Pei, J., Zhang, Y., Zhao, M., Shi, L.: Brain-inspired global-local hybrid learning towards human-like intelligence. CoRR (2020)
[27] Wu, Z., Jain, P., Wright, M.A., Mirhoseini, A., Gonzalez, J.E., Stoica, I.: Representing long-range context for graph neural networks with global attention. In: Proceedings of the 35th International Conference on Neural Information Processing Systems (2021)
[28] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems **32**, 4–24 (2021)
[29] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: Proceedings of the 35th International Conference on Machine Learning (2018)
[30] Xu, M., Wu, Y., Deng, L., Liu, F., Li, G., Pei, J.: Exploiting spiking dynamics with spatial-temporal feature normalization in graph learning. In: IJCAI (2021)

[31] Yao, M., Hu, J., Hu, T., Xu, Y., Zhou, Z., Tian, Y., XU, B., Li, G.: Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips. In: The 12nd International Conference on Learning Representations (2024)
[32] Yao, M., Hu, J., Zhou, Z., Yuan, L., Tian, Y., XU, B., Li, G.: Spike-driven transformer. In: 37th Conference on Neural Information Processing Systems (2023)
[33] Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Liu, T.Y.: Do transformers really perform badly for graph representation? In: Advances in Neural Information Processing Systems (2021)
[34] Yuan, L., Chen, Y., Wang, T., Yu, W., Shi, Y., Tay, F.E.H., Feng, J., Yan, S.: Tokens-to-token vit: Training vision transformers from scratch on imagenet. 2021 IEEE/CVF International Conference on Computer Vision (ICCV) (2021)
[35] Zhang, H., Wu, Q., Yan, J., Wipf, D., Yu, P.S.: From canonical correlation analysis to self-supervised graph neural networks. In: Advances in Neural Information Processing Systems (2021)
[36] Zhang, J., Dong, B., Zhang, H., Ding, J., Heide, F., Yin, B., Yang, X.: Spiking transformers for event-based single object tracking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022)
[37] Zhou, Z., Zhu, Y., He, C., Wang, Y., YAN, S., Tian, Y., Yuan, L.: Spikformer: When spiking neural network meets transformer. In: The 11th International Conference on Learning Representations (2023)
[38] Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., Wang, L.: Deep Graph Contrastive Representation Learning. In: ICML Workshop on Graph Representation Learning and Beyond (2020)
[39] Zhu, Z., Peng, J., Li, J., Chen, L., Yu, Q., Luo, S.: Spiking graph convolutional networks. In: IJCAI (2022)