GT-SNT: A Linear-Time Transformer for Large-Scale Graphs via Spiking Node Tokenization

Huizhe Zhang[1], **Jintang Li**[2], **Yuchang Zhu**[1], **Huazhen Zhong**[1], **Liang Chen**[1] *
[1]Sun Yat-sen University
[2]Xiamen University
zhanghzh33@mail2.sysu.edu.cn, edisonleejt@gmail.com, zhuych27@mail2.sysu.edu.cn,
zhonghzh9@mail2.sysu.edu.cn, chenliang6@mail.sysu.edu.cn

Abstract

Graph Transformers (GTs), which integrate message passing and self-attention mechanisms simultaneously, have achieved promising empirical results in graph prediction tasks. However, the design of scalable and topology-aware node tokenization has lagged behind other modalities. This gap becomes critical as the quadratic complexity of full attention renders them impractical on large-scale graphs. Recently, Spiking Neural Networks (SNNs), as brain-inspired models, provided an energy-saving scheme to convert input intensity into discrete spike-based representations through event-driven spiking neurons. Inspired by these characteristics, we propose a linear-time **G**raph **T**ransformer with **S**piking **N**ode **T**okenization (GT-SNT) for node classification. By integrating multi-step feature propagation with SNNs, spiking node tokenization generates compact, locality-aware spike count embeddings as node tokens to avoid predefined codebooks and their utilization issues. The codebook guided self-attention leverages these tokens to perform node-to-token attention for linear-time global context aggregation. In experiments, we compare GT-SNT with other state-of-the-art baselines on node classification datasets ranging from small to large. Experimental results show that GT-SNT achieves comparable performances on most datasets and reaches up to **130×** faster inference speed compared to other GTs.

Code — https://github.com/Zhhuizhe/GT-SNT

Introduction

Graph Transformers (GTs), as emerging graph representation learning paradigms, are proposed for alleviating inherent drawbacks of message passing neural networks like over-smoothing, over-squashing and local structure biases (Oono and Suzuki 2019; Topping et al. 2021). Benefiting from the multi-head attention (MHA) modules, vanilla Transformers adaptively learn the global dependencies of input sequences (Vaswani 2017). It also provides a solution for learning new topology among nodes while performing message aggregation on the graph data. Experiments demonstrate the immense potential of Transformers in handling global or long-range interactions (Rampášek et al. 2022; Bo et al. 2023). However, there is one critical drawback that Transformers with $O(N^2)$ computation complexity are prohibitive for large-scale graphs. Although empirical experiments show that designing linear-time attention in GTs can significantly reduce redundant computational cost (Wu et al. 2022, 2024), such approaches introduce potential issues related to over-globalization (Xing et al. 2024). As an orthogonal technique, encapsulating structural or semantic information into node tokens provides a promising direction that enables GTs to efficiently capture salient graph properties via complementary local and global tokens (Wang et al. 2025; Luo et al. 2024).

Recently, with the development of neuromorphic computing, Spiking Neural Networks (SNNs) offer a promising path toward energy-saving neural networks that achieve competitive performances using as few computations as possible. The defining feature of SNNs is the brain-like spiking mechanism which converts real-value signals into single-bit, sparse spiking signals based on event-driven biological neurons. The single-bit nature enables us to adopt more addition operations rather than expensive multiply-and-accumulate operations on the spiking outputs. Besides, the sparsity means spikes are cheap to store (Eshraghian et al. 2023). These delightful characteristics have prompted some studies constructing lightweight Spiking Graph Neural Networks (SGNNs) to explore spike-based representations on the graph data (Zhu et al. 2022; Li et al. 2023a,b). Despite tremendous progress in replacing artificial neurons with spiking counterparts, the broader use of SNNs for graph representation learning remains underexplored. Existing SGNNs tend to focus on their low-power advantages, while overlooking the expressive power of their sparse and discrete embeddings. SNNs naturally project continuous, high-precision inputs into low-precision, event-driven representations. This observation sparks our curiosity about an interesting research question:

Can we go beyond viewing spiking neurons merely as simple low-power units, and utilize spiking representations to build efficient tokenized Graph Transformers?

Tokenizers enable Transformer-based architectures to efficiently learn high-level representations from a small and manageable number of tokens. As shown in Figure 1, an exemplary implementation of tokenizers is VQ-VAE (Van

*Corresponding author.

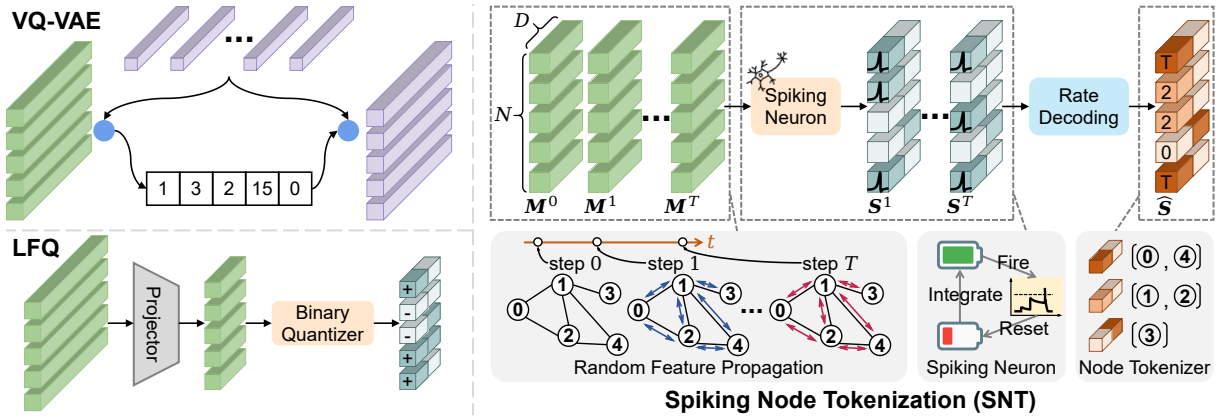

Figure 1: Frameworks of different tokenization strategies. Spiking Node Tokenization converts sequential inputs into spike count embeddings via spiking neurons, which yields a codebook-free, locality-aware node tokenization.

Den Oord, Vinyals et al. 2017), which retrieves codewords from a fixed codebook to replace original embeddings. To alleviate the issue of under-utilization, Lookup Free Quantization (LFQ) (Yu et al. 2023) proposes a learnable scalar quantization mechanism to build an implicit codebook. Inspired by existing tokenization strategies, we argue that **SNNs inherently convert sequential inputs into sparse and binary spike trains, which not only reduces energy consumption but also offers a novel sequence-to-token approach**. An energy-efficient node tokenization driven by SNNs is demonstrated in Figure 1. Spiking Node Tokenization (SNT) feeds full-precision embeddings from each propagation into spiking neurons. Spike count representations accumulated from spiking outputs represent nodes by a finite set of integer vectors, which we refer to as codewords.

Built upon the spike-driven node tokenizer, we propose GT-SNT, a linear-time **G**raph **T**ransformer based on **S**piking **N**ode **T**okenization for large-scale graphs. Specifically, GT-SNT balances local and global representations via two key components: Spiking Node Tokenization (SNT) and Codebook Guided Self-Attention (CGSA). SNT converts random feature propagation sequences into discrete, locality-aware node tokens. CGSA computes node-to-token attention scores based on generated spike count representations to capture long-range interactions in linear complexity. GT-SNT features several advantages: a) Different from existing tokenized GTs, SNT proposes a generative and adaptive tokenization scheme built upon spiking neurons to evade the issue of *codebook collapse*. b) SNT encodes graph inductive bias through well-designed sequential data. It leverages rich spiking dynamics and enables locality-aware quantization beyond simple binary quantization. c) CGSA is an efficient attention module by dynamically deriving the Key matrix from spike count representations without introducing complex machinery (e.g., distance metrics or auxiliary losses). The contributions of this paper are summarized as follows:

- We propose Spiking Node Tokenization, which links SNNs with the sequential positional encoding data to generate locality-aware node tokens. To the best of our knowledge, our work is the first to explore the usage of SNN as an energy-saving tokenizer for graph data.
- We tailor a node-to-token attention module with linear complexity, CGSA, which captures global topological information upon spike count node tokens. It provides a straightforward global aggregation guided by node tokens to balance both efficiency and expressiveness.
- We conduct a comprehensive comparison with state-of-the-art baselines across graphs with various scales. Extensive experiments show that GT-SNT achieves comparable or even superior predictive performance on most datasets. Besides, GT-SNT enjoys up to **130x** faster inference speed compared to other GT baselines.

Preliminaries

Graph Neural Network. We represent a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of edges among these nodes, $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix of the graph. Let N denote the number of nodes. We define the d-dimension nodes' attribute as $\mathbf{X} \in \mathbb{R}^{N\times d}$, which is known as the node feature matrix. For a given node $u \in \mathcal{V}$, GNN aggregates messages from its immediate neighborhood $N(u)$ and updates the node embedding h_u. This message passing process can be formulated as follows:

$$h_u^l = \text{UPDATE}(h_u^{l-1}, \text{AGGREGATE}(h_v^l, \forall v \in \mathcal{N}(u))), \tag{1}$$

where h_u^l denotes the updated embedding of node u in the l-th layer. h_u^{l-1} is the embedding from the previous layer. UPDATE and AGGREGATE can be arbitrary differentiable functions.

Self-Attention. As the most prominent component in the Transformer, the self-attention mechanism can be seen as mapping a query vector to a set of key-value vector pairs and calculating a weighted sum of value vectors as outputs. let nodes' attribute $\mathbf{X} \in \mathbb{R}^{N\times d}$ be the input to a self-attention layer. The attention function is defined as follows:

$$\text{Attn}(\mathbf{X}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d'}})\mathbf{V}, \tag{2}$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W_q}, \mathbf{K} = \mathbf{X}\mathbf{W_k}, \mathbf{V} = \mathbf{X}\mathbf{W_v}, \tag{3}$$

where Query, Key and Value are calculated by learnable projection matrices $W_q, W_k, W_v \in \mathbb{R}^{d \times d'}$. We omit the scaling factor hereafter for brevity.

Spiking Neural Network. Although electrophysiological measurements can be accurately calculated by complex conductance-based neurons, the complexity limits their widespread deployment in deep neural networks. A simplified computational unit, which is known as the Integrate-and-Fire neuron, has been proposed (Salinas and Sejnowski 2002). IF neurons have three basic characteristics: **Integrate**, **Fire** and **Reset**. Firstly, the neuron integrates synaptic inputs from other neurons or external current I to charge its cell membrane. Secondly, when the membrane potential reaches a pre-defined threshold value V_{th}, the neuron fires a spike S. Thirdly, the membrane potential of the neuron will be reset to V_{reset} after firing. The neuronal dynamics can be formulated as follows:

$$\textbf{Integrate: } V^t = \Psi(V^{t-1}, I^t) = V^{t-1} + I^t, \tag{4}$$

$$\textbf{Fire: } S^t = \Theta(V^t - V_{th}) = \begin{cases} 1, & V^t - V_{th} \geq 0 \\ 0, & \text{otherwise} \end{cases}, \tag{5}$$

$$\textbf{Reset: } V^t = V^t(1 - S^t) + V_{reset}S^t, \tag{6}$$

where V^t and I^t denote the membrane potential and input current at time step t, respectively. $\Theta(\cdot)$ denotes the fire function, implemented as a Heaviside step function. $\Psi(\cdot)$ is the membrane potential update function. Besides, there are two common IF variants, LIF and PLIF (Gerstner et al. 2014; Fang et al. 2021). The update function of these neurons can be formalized as follows:

$$\textbf{LIF: } V^t = V^{t-1} + \frac{1}{\tau}(I^t - (V^{t-1} - V_{reset})), \tag{7}$$

$$\textbf{PLIF: } V^t = V^{t-1} + \frac{1}{1 + \exp(-\beta)}(I^t - (V^{t-1} - V_{reset})), \tag{8}$$

where τ is the membrane time constant and β is a trainable parameter. They are used to regulate how fast the membrane potential decays. The rate coding scheme is widely adopted by SNNs to extract spiking dynamics from sequential data. In this paper, we transform spiking outputs into spike counts that accumulate S^t over T time steps, $\hat{S} = \sum_{t=0}^{T} S^t$. Besides, we adopt surrogate gradients during error backpropagation to address the issue of zero gradients caused by non-differentiable functions (Neftci, Mostafa, and Zenke 2019). The surrogate gradient method can be defined as $\Theta'(x) \triangleq \theta'(\alpha x)$, where α represents a smooth factor and $\theta(\cdot)$ represents a surrogate function (Neftci, Mostafa, and Zenke 2019).

GT-SNT

In this section, we comprehensively detail our approach, GT-SNT. As depicted in Figure 2, GT-SNT feeds the intermediate embeddings generated from feature propagation into the Spiking Node Tokenization (SNT) to inject graph positional information into the spike count embeddings. These codewords will guide the aggregation process in the self-attention module. Additionally, auxiliary message passing neural networks serve as encoders that encapsulate both node features and local graph topology to the attention module. In what follows, we detail the implementation of SNT. Then, we highlight how CGSA dynamically reconstructs a codebook only containing used codewords to apply node-to-token attention on large-scale graphs. Finally, we revisit the entire architecture of GT-SNT one by one.

Spiking Node Tokenization

Typically, spiking neural networks are designed for time-varying data. In order to meet the requirement for time-varying data, a common practice in spiking graph neural networks is to repeatedly pass the static graph to spiking neurons at each time step. It inevitably brings high computational and storage overheads. In this work, we collect the node embeddings over T propagation steps as the sequential inputs required for spiking neurons, which effectively reduce additional overheads. Besides, we decompose spike count embeddings accumulated from spiking outputs to dynamically reconstruct codebooks. This avoids the issue of explicitly defining the entire codebook that contains a large number of unused codewords.

Specifically, **(i)** we sample a D-dimension learnable random feature matrix $\mathbf{R} \in \mathbb{R}^{N \times D}$ from a uniform distribution, and define a propagation operator $\mathbf{P}$. Based on $\mathbf{P}$ and $\mathbf{R}$, we update and collect node embeddings $\mathbf{M} \in \mathbb{R}^{N \times D}$ over T propagation steps, $\mathcal{M} = \{\mathbf{M}^0, \mathbf{M}^1, ..., \mathbf{M}^T\}$. **(ii)** Given a SNN, it converts the above sequential node embeddings into spiking outputs $\mathcal{S} = \{\mathbf{S}^0, \mathbf{S}^1, ..., \mathbf{S}^T\}, \mathbf{S} \in \{0, 1\}^{N \times D}$. **(iii)** The spike count embeddings $\hat{\mathbf{S}} \in \mathbb{Z}^{N \times D}$ are obtained by summing $\mathbf{S}^t$ over T propagation steps. **(iv)** $\hat{\mathbf{S}}$ can be decomposed into the codebook $\mathbf{C} \in \mathbb{Z}^{B \times D}$ and the one-hot matrix $\mathbf{U} \in \{0, 1\}^{N \times B}$. The above processes are defined as:

$$\mathbf{M}^t = \mathrm{Norm}(\mathbf{P}\mathbf{M}^{t-1}), \quad \mathbf{M}^0 = \mathbf{R}, \tag{9}$$

$$\mathbf{S}^t = \Theta(\Psi(V^{t-1}, \mathbf{M}^t) - V_{th}), \tag{10}$$

$$\hat{\mathbf{S}} = \sum_{t=0}^{T} \mathbf{S}^t, \tag{11}$$

$$\mathbf{U}\mathbf{C} = \hat{\mathbf{S}}, \tag{12}$$

where $Norm(\cdot)$ normalizes output messages to the range of threshold membrane potential. $\Theta(\cdot)$ and $\Phi(\cdot)$ are membrane potential update and fire functions, respectively. We choose similar propagation operators presented in previous work (Eliasof et al. 2023). The codebook $\mathbf{C}$ is created by removing duplicate row vectors in $\hat{\mathbf{S}}$. The one-hot matrix $\mathbf{U}$ can be seen as the indices for where the codewords in $\hat{\mathbf{S}}$ ended up in $\mathbf{C}$. Hereafter, we will refer to B as the size of the reconstructed codebook.

We define a discrete latent space $\tilde{\mathcal{C}}$ of the spike count vector $\hat{s} \in \mathbb{Z}^D$, $\forall \hat{s} \in \tilde{\mathcal{C}}$. The size of $\tilde{\mathcal{C}}$ is determined by both the number of propagation steps T and the dimensionality of each random feature D, $|\tilde{\mathcal{C}}| = (T + 1)^D$. This is because

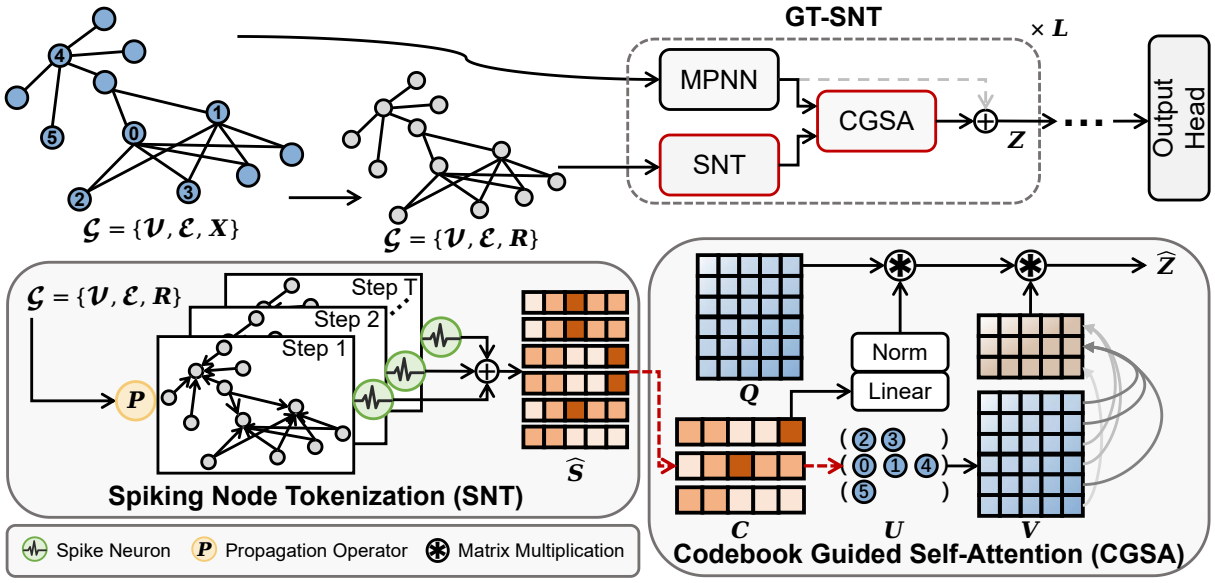

Figure 2: The overview of GT-SNT. Intuitively, the spiking node tokenization maps nodes to spike count vectors, which essentially groups nodes by their tokens. The codebook reconstructed from the spike count embeddings is fed into the codebook guided self-attention. It assists the self-attention block in calculating attention scores from nodes to grouped node sets in linear time and memory complexity.

each dimension can take $T+1$ possible spike counts (considering the case where the total number of spikes is zero) across T time steps. Different from vanilla VQ methods, which explicitly pre-define a codebook containing all latent embedding vectors for lookup, SNT dynamically maps continuous full-precision embedding to the subset of $\tilde{\mathcal{C}}$ via the SNN. It enables the size of the reconstructed codebook to be much smaller, $B \ll |\tilde{\mathcal{C}}|$, while effectively alleviating the *codebook collapse* problem.

Codebook Guided Self-Attention

After the Spiking Node Tokenization, we propose a codebook guided self-attention (CGSA) to capture long-range signals in linear complexity. Specifically, node embeddings $\mathbf{H} \in \mathbb{R}^{N \times d'}$ from auxiliary MPNNs are introduced as Query and Value. We materialize the Key $\hat{\mathbf{K}}$ based on the matrices $\mathbf{C}$ and $\mathbf{U}$. The attention functions are defined as:

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_{\mathbf{q}}, \quad \mathbf{V} = \mathbf{H}\mathbf{W}_{\mathbf{v}}, \tag{13}$$

$$\mathbf{G} = \mathrm{Norm}(\mathrm{Linear}(\mathbf{C})), \tag{14}$$

$$\hat{\mathbf{K}} = \mathbf{U}\mathbf{G}, \tag{15}$$

$$\hat{\mathbf{Z}} = \mathrm{softmax}(\mathbf{Q}\hat{\mathbf{K}}^T)\mathbf{V}, \tag{16}$$

where d' denotes the dimension of intermediate embeddings. In CGSA, $\hat{\mathbf{K}}$ is calculated based on the matrix multiplication between $\mathbf{G}$ and $\mathbf{U}$, rather than performing the nearest neighbor look-up on the entire codebook. It avoids the severe reliance on complex components like distance measures or auxiliary losses. Derived from (Lingle 2023), the attention weights in Eq 16 can be further factored:

$$\begin{aligned}\hat{\mathbf{Z}} &= \mathrm{softmax}(\mathbf{Q}\hat{\mathbf{K}}^T)\mathbf{V} \\ &= \mathrm{softmax}(\mathbf{Q}(\mathbf{U}\mathbf{G})^T)\mathbf{V} \\ &= \mathrm{Diag}^{-1}(\exp(\mathbf{Q}\mathbf{C}^T)\mathbf{U}^T\mathbf{1})\exp(\mathbf{Q}\mathbf{C}^T)\mathbf{U}^T\mathbf{V}\end{aligned} \tag{17}$$

where $\mathbf{1} \in \mathbb{R}^N$. $\mathbf{U}^T\mathbf{1} = \{n_b\}_b^B \in \mathbb{Z}^+$ denotes the number of spike count embeddings in $\hat{\mathbf{S}}$ mapped to the same codewords. The complexity of CGSA is $O(NBd_v)$, where $B \ll N$. It can be considered that the computational overhead of CGSA grows linearly with the number of nodes. To avoid generating an excessively large codebook in the initial phase of learning, we perform a truncation strategy. We rank n_b from high to low and select the top B_{max} to generate a truncated codebook, which ensures the training efficiency of GT-SNT.

Overall Framework

As shown in Figure 2, the overview of GT-SNT includes four modules: SNT, auxiliary MPNN, CGSA and a classification head (CH). In the SNT, we construct learnable random features and spiking neurons for each layer. By defining a shared propagation operator, messages among nodes are collected and transformed into spike count embeddings $\hat{\mathbf{S}}$. Then auxiliary MPNNs as encoders generate node embeddings with semantic and local neighborhood information. In the CGSA, the reconstructed codebook $\mathbf{C}$, codeword indices $\mathbf{U}$ and node embeddings $\mathbf{H}$ are fed into a linear-time self-attention to capture global topological information. Different from the vanilla Transformer, CGSA employs global node-to-token attention to actively introduce graph inductive

bias. These four parts can be written as follows:

$$\mathbf{U}^l, \mathbf{C}^l = \mathrm{SNT}^l(\mathbf{A}), \tag{18}$$

$$\mathbf{H}^l = \mathrm{MPNN}^l(\mathbf{Z}^{l-1}, \mathbf{A}), \tag{19}$$

$$\hat{\mathbf{Z}}^l = \mathrm{CGSA}^l(\mathbf{U}^l, \mathbf{C}^l, \mathbf{H}^l), \tag{20}$$

$$\mathbf{Z}^l = \mathrm{Linear}(\hat{\mathbf{Z}}^l) + \mathbf{H}^l, \tag{21}$$

$$\mathbf{Y} = \mathrm{CH}(\mathbf{Z}^L), \tag{22}$$

where L is the number of layers. We choose a simple fully connected layer as the classification head. The auxiliary MPNNs are implemented as a single-layer GCN without normalization. It has been observed that projection blocks, including Multilayer Perceptrons (MLPs) and normalization layers, exacerbate the overfitting problem on large-scale graphs for vanilla Transformers. Therefore, we discard projection layers and retain only the self-attention module and the residual connection (He et al. 2016).

Experiments

Comparison with Existing Models

In this section, we conduct the experimental evaluation to show the effectiveness of GT-SNT on nine node classification datasets, including three citation networks (Kipf and Welling 2016), two co-purchase networks (Shchur et al. 2018), two heterophilic graphs (Wu et al. 2024) and two OGB graphs (Hu et al. 2020). A head-to-head comparison is conducted with state-of-the-art GNNs and GTs, based on their architectural designs. As shown in Table 1, these baselines fall into three categories: spike-based methods, Graph Transformer framework and vector quantization-based methods. The hyperparameter search strategy is deployed on both GT-SNT and other baselines to get the optimal combinations of parameters. We perform all models on each dataset 5 times with different random seeds to report the mean and standard deviation of metrics. All experiments are conducted on a single NVIDIA RTX 4090 GPU.

Overall performance. The experimental results are demonstrated in Table 2. As shown in the table, our method achieves promising performance on all datasets. This is a significant advancement considering the information loss caused by low-precision spike count embeddings. Specifically, GT-SNT achieves predictive performances on par or even better than high-precision GT baselines. **In general, our method shows slight improvements on 5 out of 9 datasets**. GT-SNT effectively injects the graph inductive bias into the self-attention block while capturing long-range interactions, resulting in strong expressive power. Besides, we observe that GT-SNT has competitive performances on two heterophily datasets, particularly Actor. It suggests that GT-SNT can also maintain strong expressive power on heterophilic graphs. **GT-SNT outperforms other spike-based baselines on all datasets, which achieves an average improvement of 3.9%.** Constructing the input sequence from propagation steps rather than repeating graphs multiple times endows GT-SNT with desirable scalability. In addition, GT-SNT achieves a better trade-off between latent space compression and information retention by introducing spiking neurons as the node tokenizer instead of artificial neuron substitutes.

Models	Components		
	SP	GT	VQ
SpikingGCN(Zhu et al. 2022)	✓	-	-
SpikeNet(Li et al. 2023a)	✓	-	-
SpikeGCL(Li et al. 2023b)	✓	-	-
SpikeGT(Sun et al. 2024)	✓	✓	-
NAGphormer(Chen et al. 2022)	-	✓	-
NodeFormer(Wu et al. 2022)	-	✓	-
SGFormer(Wu et al. 2024)	-	✓	-
GOAT(Kong et al. 2023)	-	✓	✓
VQGraph(Yang et al. 2024)	-	-	✓
GT-SNT	✓	✓	✓

Table 1: Comparison of Graph Transformers and Graph Neural Networks w.r.t. required components (**SP**: spike-based, **GT**: Graph Transformer framework, **VQ**: vector quantization-based).

Characteristics of Spiking Node Tokenization

To thoroughly analyze the spiking node tokenization, we conduct a series of experiments on the GT-SNT. We track the metrics including Codebook Usage and Accuracy to explore the following questions: **(i)** How does the size of the latent embedding space affect GT-SNT? **(ii)** Is the spiking node tokenization a more efficient tokenization alternative?

The influence of $|\tilde{\mathcal{C}}|$. As aforementioned above, the number of propagation steps T and the random feature dimension D determine the size of the latent space. In Figure 3, we evaluate the performances of GT-SNT under different combinations between D and T. The visualization results show that an excessively small latent space size may impair the performance of GT-SNT. It generally improves accuracy as the latent space size increases. Essentially, $|\tilde{\mathcal{C}}|$ determines the upper bound of information the reconstructed codebook can store. When $|\tilde{\mathcal{C}}|$ is much smaller than the number of nodes, indiscriminately mapping a large number of nodes to the same codeword will bring significant information loss. Besides, we observe the diminishing gains from latent space size scaling, especially on small-scale graphs. We believe that node embeddings merely are transformed into low-precision counterparts rather than being compressed appropriately when the feature dimensionality D is too large. The propagation step T correlates with the complexity of spiking patterns. As depicted in Figure 3, it suggests that $T > 6$ may lead to subpar performance. Notably, GT-SNT outperforms most spike-based baselines even under the worst parameter combinations. It demonstrates that GT-SNT evades the difficulty of codebook learning and provides a robust, easy-to-use node tokenization approach.

Codebook Usage. For exploring the *codebook collapse* problem in VQ-based graph models, we track the fraction of codewords that are used at least once after the training phase. The codebook usages between GT-SNT and the other

Models	Cora	CiteSeer	PubMed	Co-CS	Co-Physics	Actor	Deezer	arXiv	Products
#nodes	2,708	3,327	19,717	18,333	34,493	7,600	28,281	169,343	2,449,029
#edges	10,556	9,104	88,648	163,788	495,924	30,019	185,504	1,166,243	61,859,140
GCN	$81.6_{\pm 0.4}$	$71.6_{\pm 0.4}$	$78.8_{\pm 0.6}$	$92.5_{\pm 0.4}$	$95.7_{\pm 0.5}$	$30.1_{\pm 0.2}$	$62.7_{\pm 0.7}$	$70.4_{\pm 0.3}$	$75.7_{\pm 0.1}$
GAT	$83.0_{\pm 0.7}$	$72.1_{\pm 1.1}$	$79.0_{\pm 0.4}$	$92.3_{\pm 0.2}$	$95.4_{\pm 0.3}$	$29.8_{\pm 0.6}$	$61.7_{\pm 0.8}$	$70.6_{\pm 0.3}$	OOM
SGC	$80.1_{\pm 0.2}$	$71.9_{\pm 0.1}$	$78.7_{\pm 0.1}$	$90.3_{\pm 0.9}$	$93.2_{\pm 0.5}$	$27.0_{\pm 0.9}$	$62.3_{\pm 0.4}$	$68.7_{\pm 0.1}$	$74.2_{\pm 0.1}$
VQGraph	$81.1_{\pm 1.2}$	**$74.5_{\pm 1.9}$**	$77.1_{\pm 3.0}$	$93.3_{\pm 0.1}$	$95.0_{\pm 0.1}$	**$38.7_{\pm 1.6}$**	$65.1_{\pm 0.2}$	**$72.4_{\pm 0.2}$**	**$78.3_{\pm 0.1}$**
SpikingGCN	$79.1_{\pm 0.5}$	$62.9_{\pm 0.1}$	$78.6_{\pm 0.4}$	$92.6_{\pm 0.3}$	$94.3_{\pm 0.1}$	$26.8_{\pm 0.1}$	$58.2_{\pm 0.3}$	$55.8_{\pm 0.7}$	OOM
SpikeNet	$78.4_{\pm 0.7}$	$64.3_{\pm 0.8}$	$79.1_{\pm 0.5}$	$93.0_{\pm 0.1}$	$95.8_{\pm 0.7}$	$36.2_{\pm 0.9}$	$65.0_{\pm 0.2}$	$66.8_{\pm 0.1}$	$74.3_{\pm 0.4}$
SpikeGCL	$79.8_{\pm 0.7}$	$64.9_{\pm 0.2}$	$79.4_{\pm 0.8}$	$92.8_{\pm 0.1}$	$95.2_{\pm 0.6}$	$30.3_{\pm 0.5}$	$65.0_{\pm 1.1}$	$70.9_{\pm 0.1}$	OOM
SpikeGT	$82.0_{\pm 0.7}$	$70.5_{\pm 0.6}$	$71.1_{\pm 0.4}$	$92.1_{\pm 0.8}$	$95.7_{\pm 0.3}$	$36.0_{\pm 0.5}$	$65.6_{\pm 0.2}$	$70.2_{\pm 0.9}$	OOM
NAGphormer	$79.9_{\pm 0.1}$	$68.8_{\pm 0.2}$	**$80.3_{\pm 0.9}$**	$93.1_{\pm 0.5}$	$95.7_{\pm 0.7}$	$33.0_{\pm 0.9}$	$64.4_{\pm 0.6}$	$70.4_{\pm 0.3}$	$73.3_{\pm 0.7}$
GOAT	$78.6_{\pm 0.5}$	$68.4_{\pm 0.7}$	$78.1_{\pm 0.5}$	**$93.5_{\pm 0.6}$**	$95.4_{\pm 0.2}$	$37.5_{\pm 0.7}$	$65.1_{\pm 0.3}$	**$72.4_{\pm 0.4}$**	**$82.0_{\pm 0.4}$**
NodeFormer	$82.2_{\pm 0.9}$	$72.5_{\pm 1.1}$	$79.9_{\pm 1.0}$	$92.9_{\pm 0.1}$	$95.4_{\pm 0.1}$	$36.9_{\pm 1.0}$	**$66.4_{\pm 0.7}$**	$59.9_{\pm 0.4}$	$72.9_{\pm 0.1}$
SGFormer	**$84.5_{\pm 0.8}$**	$72.6_{\pm 0.2}$	**$80.3_{\pm 0.6}$**	$91.8_{\pm 0.2}$	**$95.9_{\pm 0.8}$**	$37.9_{\pm 1.1}$	**$67.1_{\pm 1.1}$**	**$72.6_{\pm 0.1}$**	$72.6_{\pm 1.2}$
GT-SNT	**$84.7_{\pm 0.8}$**	**$74.0_{\pm 0.5}$**	**$80.6_{\pm 0.4}$**	**$93.7_{\pm 0.4}$**	**$96.2_{\pm 0.0}$**	**$39.1_{\pm 0.2}$**	$65.7_{\pm 0.1}$	**$72.4_{\pm 0.3}$**	$74.8_{\pm 0.4}$

Table 2: Classification accuracy(%) on nine datasets. Highlighted are the top **first**, **second** results.

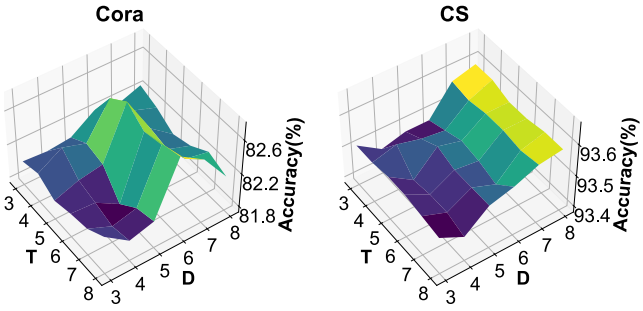

Figure 3: Influences of random feature dimensionality D and propagation step T on Cora and CS.

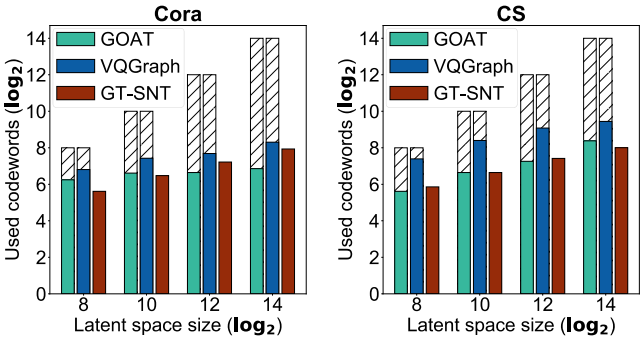

Figure 4: Codebook usage under four different latent space sizes on Cora and CS datasets. The solid-colored bar represents the number of used codewords, while the hatched bar denotes the size of the pre-defined codebook.

VQ-based graph methods are shown in Figure 4. In previous works, the pre-defined codebook is used to represent the entire discrete latent space. We select four combinations of D and T ($T = 3/D = 4$, $T = 3/D = 5$, $T = 3/D = 6$, $T = 3/D = 7$) to match the latent space sizes (2^8, 2^{10}, 2^{12}, 2^{14}). The results demonstrate that those methods using pre-defined codebooks suffer from the serious issue of *codebook collapse*. As the pre-defined codebook size increases, the codebook usage decreases. For GOAT, the average codebook usages are 10.6% and 8.7% on Cora and CS datasets. The codebook usages in VQGraph are slightly higher, which achieve 16.9% and 29.0%. When the latent space size exceeds 2^{10}, the usages of both methods drop below 50%. It reveals that these VQ-based graph learning methods struggle with efficiently using large pre-defined codebooks. SNT provides a generative solution, which dynamically selects the subset from a discrete spike count embedding space to reconstruct the codebook. It brings 100% codebook usage. Although the number of used codewords in GT-SNT is slightly larger than that of vanilla VQ counterparts in some cases, it is still significantly smaller than the pre-defined codebooks. In a nutshell, SNT achieves better codebook utilization by viewing spiking neurons as a sequence-to-token approach.

Energy Efficiency Analysis

To verify the efficiency of GT-SNT, we conduct energy efficiency analysis using the following metrics: peak memory usage during training, inference latency and theoretical energy consumption. The theoretical energy consumption estimation is derived from (Yao et al. 2024). The results are presented in Table 3.

The results show that GT-SNT achieves the lowest inference latency across all datasets. **Compared to other GTs, GT-SNT with better performance achieves up to 130x lower inference latency.** Most VQ-based methods tend to retrieve and replace node representations with the closest codewords during the inference phase. In SNT, trained spiking neurons directly convert input features into codewords, which reduces the computational cost brought by retrieval. Integrating the reconstructed codebook in SNT with linear-time global attention brings a significant improvement in in-

Datasets	Metrics	NAGphormer	GOAT	NodeFormer	SGFormer	SpikeGT	Avg	GT-SNT
CS	Latency↓	0.70	5.02	0.05	**0.01**	**0.03**	1.16	**0.01** (×116)
	Memory↓	3400	12490	2822	**1662**	8542	5783	**1638**
	Energy↓	0.82	1.21	**0.21**	0.35	0.59	0.64	**0.16**
Physics	Latency↓	1.79	10.98	0.14	**0.02**	**0.08**	2.60	**0.02** (×130)
	Memory↓	13628	22776	7624	**2944**	16414	12677	**3036**
	Energy↓	1.86	2.35	**0.46**	0.78	1.35	1.36	**0.37**
arXiv	Latency↓	0.78	28.27	1.17	**0.10**	0.30	6.12	**0.08** (×77)
	Memory↓	10450	21146	11988	**6386**	22654	14525	**7132**
	Energy↓	1.12	9.92	0.63	0.57	**0.13**	2.47	**0.18**
Products	Latency↓	25.74	2416.84	41.78	**24.34**	-	627.18	**20.83** (×30)
	Memory↓	**7470**	21974	10500	**934**	-	10220	13494
	Energy↓	16.06	143.80	10.05	**8.07**	-	44.50	**3.69**

Table 3: The maximum memory usage (MB), theoretical energy consumption (J) and inference latency (s) of various GT methods. The speedup over average inference latency is underlined.

ference speed. For large-scale graphs with high feature dimensionality, repeating the graph to generate sequential inputs leads to high energy consumption. The results show that previous spike-based GT is hard to manifest its superiority in energy consumption on CS and Physics. Through generating low-dimensional random features and collecting sequential embeddings from propagation steps, **GT-SNT outperforms another spike-based GT in memory usage and latency across all datasets while maintaining acceptable theoretical energy consumption.**

Ablation Study

In this section, we conduct ablation studies to explore the influences of different components on predictive performances. We construct 8 baselines by combining two classic linear-time attention modules (Performer (Choromanski et al. 2020) and Linformer (Wang et al. 2020)), two spiking neurons (IF and LIF), two normalization algorithms (LayerNorm (Ba, Kiros, and Hinton 2016) and STFNorm (Xu et al. 2021)) and two VQ modules (VQ-VAE (Van Den Oord, Vinyals et al. 2017) and FSQ (Mentzer et al. 2023)).

The experimental results are demonstrated in Table 4. Although incorporating extra positional encodings enables Performer and Linformer to handle graph prediction tasks, they struggle to achieve good predictive performance on large-scale graphs like ogbn-arxiv. In GT-SNT, the CGSA actively introduces the global positional encoding during attention score calculation. It suggests that developing topology-aware tokenized Transformers is a promising direction for scaling GTs on large-scale graphs. The choice of spiking neurons also affects the predictive performances of GT-SNT. PLIF models with learnable membrane time constants and synaptic weights achieve better performances in most cases. These neurons effectively improve the flexibility of SNT. In addition, the well-designed normalization algorithm for spiking neurons, STFNorm outperforms the LayerNorm algorithm across all datasets. For complex tasks, GT-SNT with dynamic learning and adaptive codebook granularity surpasses the simple quantizers such as FSQ. Benefitting from diverse spiking dynamics and finer quantization, GT-SNT outperforms both FSQ and full-precision VQ-VAE.

Models	PubMed	CS	Physics	ogbn-arxiv
+Performer	$80.2_{\pm 0.2}$	$93.1_{\pm 0.4}$	$95.8_{\pm 0.1}$	$71.2_{\pm 0.1}$
+Linformer	$79.6_{\pm 1.0}$	$92.6_{\pm 0.5}$	$95.5_{\pm 0.1}$	$65.2_{\pm 1.3}$
+IF	$81.6_{\pm 1.2}$	$92.8_{\pm 0.1}$	$96.0_{\pm 0.4}$	$71.0_{\pm 0.5}$
+LIF	$79.6_{\pm 0.7}$	$92.8_{\pm 0.0}$	$96.1_{\pm 0.2}$	$72.1_{\pm 0.2}$
+LayerNorm	$78.9_{\pm 1.3}$	$90.3_{\pm 0.6}$	$95.4_{\pm 0.4}$	$71.2_{\pm 0.2}$
+STFNorm	**82.6**$_{\pm 0.2}$	$92.3_{\pm 0.4}$	**96.5**$_{\pm 0.5}$	**72.4**$_{\pm 0.7}$
+VQ	$79.8_{\pm 0.4}$	$93.2_{\pm 0.3}$	$95.0_{\pm 0.4}$	$70.7_{\pm 0.7}$
+FSQ	$78.4_{\pm 0.9}$	**93.7**$_{\pm 0.2}$	$95.8_{\pm 0.1}$	$71.6_{\pm 0.2}$
GT-SNT	$80.6_{\pm 0.5}$	**93.7**$_{\pm 0.4}$	$96.2_{\pm 0.0}$	**72.4**$_{\pm 0.3}$

Table 4: Ablation studies on PubMed, CS, Physics and ogbn-arxiv datasets. And $+x$ means replacing the original component in GT-SNT with x.

Conclusion

In this study, we propose GT-SNT, a linear-time Graph Transformer via spiking node tokenization. We find that positional encoding patterns of different nodes can be encoded into the same codewords. Inspired by the observation, GT-SNT not only considers spiking neurons as the low-power units, but also incorporates SNNs as a learnable tokenizer into Graph Transformers. It enables GT-SNT to achieve faster inference speed and better predictive performance. Spiking node tokenization that dynamically creates a spike count-based codebook paves a different way to address issues within current VQ-based graph models. We believe that our work holds great promise for the development of brain-inspired neural networks, which bridge the gap between existing SNNs and node tokenization. We hope it will inspire further research into energy-saving Graph Transformers.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant 2022YFF0902500, in part by the Guangdong Basic and Applied Basic Research Foundation, China under Grant 2023A1515011050, in part by Shenzhen Science and Technology Program under Grant KJZD20231023094501003, and in part by Tencent AI Lab under Grant RBFR2024004.

References

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. arXiv:1607.06450.

Bo, D.; Shi, C.; Wang, L.; and Liao, R. 2023. Specformer: Spectral graph neural networks meet transformers. *arXiv preprint arXiv:2303.01028*.

Chen, J.; Gao, K.; Li, G.; and He, K. 2022. NAGphormer: A tokenized graph transformer for node classification in large graphs. *arXiv preprint arXiv:2206.04910*.

Choromanski, K.; Likhosherstov, V.; Dohan, D.; Song, X.; Gane, A.; Sarlos, T.; Hawkins, P.; Davis, J.; Mohiuddin, A.; Kaiser, L.; et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Eliasof, M.; Frasca, F.; Bevilacqua, B.; Treister, E.; Chechik, G.; and Maron, H. 2023. Graph positional encoding via random feature propagation. In *International Conference on Machine Learning*, 9202–9223. PMLR.

Eshraghian, J. K.; Ward, M.; Neftci, E. O.; Wang, X.; Lenz, G.; Dwivedi, G.; Bennamoun, M.; Jeong, D. S.; and Lu, W. D. 2023. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*.

Fang, W.; Yu, Z.; Chen, Y.; Masquelier, T.; Huang, T.; and Tian, Y. 2021. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2661–2671.

Gerstner, W.; Kistler, W. M.; Naud, R.; and Paninski, L. 2014. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33: 22118–22133.

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

Kong, K.; Chen, J.; Kirchenbauer, J.; Ni, R.; Bruss, C. B.; and Goldstein, T. 2023. GOAT: A global transformer on large-scale graphs. In *International Conference on Machine Learning*, 17375–17390. PMLR.

Li, J.; Yu, Z.; Zhu, Z.; Chen, L.; Yu, Q.; Zheng, Z.; Tian, S.; Wu, R.; and Meng, C. 2023a. Scaling up dynamic graph representation learning via spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 8588–8596.

Li, J.; Zhang, H.; Wu, R.; Zhu, Z.; Wang, B.; Meng, C.; Zheng, Z.; and Chen, L. 2023b. A graph is worth 1-bit spikes: When graph contrastive learning meets spiking neural networks. *arXiv preprint arXiv:2305.19306*.

Lingle, L. D. 2023. Transformer-vq: Linear-time transformers via vector quantization. *arXiv preprint arXiv:2309.16354*.

Luo, Y.; Li, H.; Liu, Q.; Shi, L.; and Wu, X.-M. 2024. Node Identifiers: Compact, Discrete Representations for Efficient Graph Learning. arXiv:2405.16435.

Mentzer, F.; Minnen, D.; Agustsson, E.; and Tschannen, M. 2023. Finite scalar quantization: Vq-vae made simple. *arXiv preprint arXiv:2309.15505*.

Neftci, E. O.; Mostafa, H.; and Zenke, F. 2019. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6): 51–63.

Oono, K.; and Suzuki, T. 2019. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*.

Rampášek, L.; Galkin, M.; Dwivedi, V. P.; Luu, A. T.; Wolf, G.; and Beaini, D. 2022. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35: 14501–14515.

Salinas, E.; and Sejnowski, T. J. 2002. Integrate-and-fire neurons driven by correlated stochastic input. *Neural computation*, 14(9): 2111–2155.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

Sun, Y.; Zhu, D.; Wang, Y.; Tian, Z.; Cao, N.; and O'Hared, G. 2024. SpikeGraphormer: A High-Performance Graph Transformer with Spiking Graph Attention. *arXiv preprint arXiv:2403.15480*.

Topping, J.; Di Giovanni, F.; Chamberlain, B. P.; Dong, X.; and Bronstein, M. M. 2021. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*.

Van Den Oord, A.; Vinyals, O.; et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Vaswani, A. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Wang, L.; Hassani, K.; Zhang, S.; Fu, D.; Yuan, B.; Cong, W.; Hua, Z.; Wu, H.; Yao, N.; and Long, B. 2025. Learning Graph Quantized Tokenizers. arXiv:2410.13798.

Wang, S.; Li, B. Z.; Khabsa, M.; Fang, H.; and Ma, H. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Wu, Q.; Yang, K.; Zhang, H.; Wipf, D.; and Yan, J. 2024. SGFormer: Single-Layer Graph Transformers with Approximation-Free Linear Complexity. *arXiv preprint arXiv:2409.09007*.

Wu, Q.; Zhao, W.; Li, Z.; Wipf, D. P.; and Yan, J. 2022. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35: 27387–27401.

Xing, Y.; Wang, X.; Li, Y.; Huang, H.; and Shi, C. 2024. Less is more: on the over-globalizing problem in graph transformers. *arXiv preprint arXiv:2405.01102*.

Xu, M.; Wu, Y.; Deng, L.; Liu, F.; Li, G.; and Pei, J. 2021. Exploiting spiking dynamics with spatial-temporal feature normalization in graph learning. *arXiv preprint arXiv:2107.06865*.

Yang, L.; Tian, Y.; Xu, M.; Liu, Z.; Hong, S.; Qu, W.; Zhang, W.; Bin, C.; Zhang, M.; and Leskovec, J. 2024. VQGraph: Rethinking graph representation space for bridging GNNs and MLPs. In *The Twelfth International Conference on Learning Representations*.

Yao, M.; Hu, J.; Hu, T.; Xu, Y.; Zhou, Z.; Tian, Y.; Xu, B.; and Li, G. 2024. Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips. *arXiv preprint arXiv:2404.03663*.

Yu, L.; Lezama, J.; Gundavarapu, N. B.; Versari, L.; Sohn, K.; Minnen, D.; Cheng, Y.; Birodkar, V.; Gupta, A.; Gu, X.; et al. 2023. Language Model Beats Diffusion–Tokenizer is Key to Visual Generation. *arXiv preprint arXiv:2310.05737*.

Zhu, Z.; Peng, J.; Li, J.; Chen, L.; Yu, Q.; and Luo, S. 2022. Spiking graph convolutional networks. *arXiv preprint arXiv:2205.02767*.