Preserving Spatial-Temporal Relationship with Adaptive Node Sampling in Hierarchical Dynamic Graph Transformers

Thi Linh Hoang [*] HTLINH83@GMAIL.COM
Singapore Management University, Singapore

Tuan Dung Pham TPHAM01@QUB.AC.UK
Son Thai Mai THAISON.MAI@QUB.AC.UK
Queen's University Belfast, United Kingdom

Viet Cuong Ta[†] CUONGTV@VNU.EDU.VN
VNU University of Engineering and Technology, Hanoi, Vietnam

Editors: Vu Nguyen and Hsuan-Tien Lin

Abstract

Dynamic Graph Transformers (DGTs) have demonstrated remarkable performance in various applications, such as social networks, traffic forecasting, and recommendation systems. Despite their effectiveness in capturing long-range dependencies, training DGTs for large graphs remains a challenge. Mini-batch training is usually used to alleviate this challenge but this approach often fails to capture complex dependencies or sacrifice performance. To deal with the above problems, we propose the Adaptive Node Sampling in Hierarchical Dynamic Graph Transformers (ASH-DGT) architecture that focuses on sampling the set of suitable nodes preserving spatial-temporal relationships in the dynamic graph for training DGTs. Unlike previous methods that use random sampling or structural sampling, our motivation is that the contribution of nodes to learning performance can be time-sensitive, while we still care about spatial correlation in the dynamic graph with consideration to the global and local structure of the graph. Through extensive evaluations on popular real-world datasets for node classification and link prediction, ASH-DGT consistently outperforms multiple state-of-the-art methods, achieving both higher accuracy and significant improvements in training efficiency.

Keywords: Graph Transformer, Laplacian Positional Encoding, Structural Encoding, Adaptive Sampling.

1. Introduction

In recent years, GNNs have attracted a certain amount of attention from researchers and have shown impressive results for problems where data can be represented as a graph, such as recommendation system He et al. (2020), social network Hoang et al. (2021), and traffic networks Kong et al. (2024). Traditionally, these graphs are static with a fixed number of nodes and edges. However, in real-world applications, many graphs are dynamic, in which their entities and interactions continuously evolve over time Skarding et al. (2021).

Existing works for dynamic graph modeling are divided into two categories: discrete-time approaches Goyal and Ferrara (2018); Hajiramezanali et al. (2019); Pareja et al. (2020); Sankar et al. (2020); Jiang et al. (2022), and continuous-time approaches Xu et al. (2020);

[*] This work was partly done while at VNU-UET
[†] Corresponding author.

Kumar et al. (2019); Trivedi et al. (2019); Rossi et al. (2020). Discrete-time methods improve the snapshot-based approach that utilizes learning embedding by the GNN-base models by adding temporal relations to the node representation, but there are issues in learning the fine-grained temporal structure of the dynamic graph. Continuous-time methods avoid these issues by seeing the dynamic graph as a sequence of nodes' interactions with a timestamp. Recently, transformer-based approaches have been employed in continuous-time dynamic graph modeling due to their ability to capture long-range dependencies and complex temporal patterns. For instance, DyGFormer Yu et al. (2023) leverages a neighbor co-occurrence encoding scheme and a patching technique to capture node correlations and long-term temporal dependencies from first-hop interactions. Another work on Spatial-Temporal Graph Transformers Kong et al. (2024) has shown promise in predicting traffic flow patterns in transportation networks. These advancements highlight the potential of transformer-based approaches for dynamic graph learning. However, these transformer-based methods face significant challenges in scaling to large graphs and maintaining spatial-temporal relationships due to the computational complexity of the self-attention mechanism and the need for extensive memory resources.

To reduce the computation requirements, sampling-based techniques can be used to select the relevant neighbors to attend to a node's local information. For dynamic graph learning, previous work Rossi et al. (2020) uses uniform sampling from neighbors or samples from the most recent interactions. However, these methods have significant disadvantages. Uniform sampling can lead to the selection of less informative or irrelevant neighbors, thereby diluting the quality of the learned embeddings. While sampling only from the most recent interactions may overlook important historical connections, thereby failing to capture long-term dependencies. Other approaches Trivedi et al. (2019); Ahmed et al. (2016) employ random walk models to sample subgraphs, capturing both local and global structures, but they often require extensive computational resources and may overlook transient patterns crucial in dynamic settings.

Graph coarsening is a technique used to reduce the size of a graph by merging nodes and edges while preserving the essential structural properties of the original graph. This process creates a hierarchical representation of the graph, which can significantly enhance the efficiency and scalability of graph-based models. Hierarchical approaches enable multi-scale learning, where features are progressively aggregated from finer to coarser levels, thus providing a comprehensive understanding of the graph's topology. Recent studies, such as Cangea et al. (2018) and Ying et al. (2018) have demonstrated the benefits of hierarchical graph representations in improving model performance and computational efficiency.

In this work, we enhance the general learning framework on dynamic graphs with the transformer-based model by incorporating hierarchical information into the node sampling. introduce an approach called $\underline{\text{A}}$daptive Node $\underline{\text{S}}$ampling in $\underline{\text{H}}$ierarchical $\underline{\text{D}}$ynamic $\underline{\text{G}}$raph $\underline{\text{T}}$ransformers (ASH-DGT) that addresses these challenges by introducing an adaptive node sampling mechanism that efficiently preserves spatial-temporal relationships while significantly reducing computational overhead. By dynamically selecting the most relevant nodes and operating on a coarser graph representation for hierarchy attention, our approach captures crucial spatial-temporal contexts and long-range dependencies, leading to more accurate and scalable learning in dynamic graph scenarios. We evaluate our proposed ASH-DGT model by extensive testing on 6 dynamic graph datasets with two downstream tasks, node classification and link prediction. The results highlight that our model increases the performance 2-3% across different benchmarks while the training computation complexity reduces significantly. Further testing with the node sampling module and graph coarsening module reveals that our model can balance performance and computation speed and capture the temporal information effectively.

2. Background

2.1. Dynamic Graph Modeling

We denote a dynamic graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{X})$ with a set of nodes $\mathcal{V}$, edges $\mathcal{E}$, times $\mathcal{T}$ and node features $\mathcal{X}$. $\mathcal{G}$ can capature from sequence of non-decreasing chronological interactions $G = \{(u_i, v_i, t_i)\}$ with $0 \leq t_i \leq T$, where $u_i, v_i \in \mathcal{V}$ denote the source node and destination node of the i-th link at timestamp t_i. Each node $u \in \mathcal{V}$ can be associated with node feature $\boldsymbol{x}_u \in \mathbb{R}^{d_V}$, and each interaction (u, v, t) has edge features $\boldsymbol{e}^t_{u,v} \in \mathbb{R}^{d_E}$, where d_V and d_E denote the dimensions of the node features and edge features. If the graph is non-attributed, we simply set the node features and link features to zero vectors, i.e., $\boldsymbol{x}_u = 0$ and $\boldsymbol{e}^t_{u,v} = 0$.

Given the source node u, destination node v, timestamp t, and historical interactions before t, i.e., $\{(u', v', t') \mid t' < t\}$, representation learning on dynamic graph aims to design a model to learn time-aware representations $\boldsymbol{h}^t_u \in \mathbb{R}^d$ and $\boldsymbol{h}^t_v \in \mathbb{R}^d$ for u and v with d as the dimension. We validate the effectiveness of the learned representations via two classic tasks in dynamic graph learning: (i) dynamic link prediction, which predicts whether u and v are connected at t; (ii) dynamic node classification, which infers the state of u or v at t.

2.2. Transformer Architecture

The Transformer model, introduced by Vaswani et al. (2017), has revolutionized the field of deep learning. At the core of the Transformer is the self-attention mechanism, which allows the model to dynamically weigh the relevance of different elements in the input sequence. Formally, given a sequence of input embeddings $X = [x_1, x_2, \ldots, x_n]$, the self-attention mechanism produces a set of output vectors $Z = [z_1, z_2, \ldots, z_n]$ where each z_i is computed as a weighted sum of linearly transformed input vectors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

Here, Q, K, and V are matrices of queries, keys, and values derived from the input embeddings, and d_k is the dimension of the keys. The Transformer architecture also incorporates multi-head attention, where multiple self-attention mechanisms (heads) are applied in parallel. This allows the model to capture diverse aspects of relationships within the data and each head i computes the self-attention output using different learned projections.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{2}$$

3. Our proposed method

In this section, we present our proposed approach ASH-DGT, which is designed with two main modules, the Graph Hierarchy module and the Adaptive Node Sampling module

(Figure 1). At each update time step, both modules are used to extract relevant information from the graph which is represented under a node sequence to improve the target node embedding. Firstly, the Graph Hierarchy module coarsens the graph to a smaller one, the results of this module are super nodes that represent the global structure of the graph. Then, the Adaptive Node Sampling selects neighbor nodes of each target node instead of relying on the full set of the target node's neighbors using previous attention, and embedding scores. Inspired by the work of Hwang et al. (2022) on enhancing expressiveness in graph neural networks, we add virtual nodes in dynamic graph transformers (DGTs) to augment their ability to capture global and time-varying structural information. Then, the node sequence constructed by nodes results in adaptive sampling, and super-nodes from the coarser graph are fed into Transformer layers for capturing long-term dependencies. As the self-attention operator of the Transformer needs to encode the structural information of the node sequence, our proposed ASH-DGT employs a further Positional-Temporal Encoding module to extract the global position information and temporal information into the raw features of each node.

3.1. Graph Hierarchy and Adaptive Node Sampling

3.1.1. Graph Hierarchy

In the Graph Hierarchy module, we use the graph coarsening technique introduced by Loukas (2019); Ron et al. (2011), it summarize a larger graph into a smaller graph while preserving the properties of the original graph. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the graph coarsening algorithm aims to construct an appropriate coarser graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ that contains certain properties of $\mathcal{G}$. $\mathcal{G}'$ is obtained from the original graph by calculating a partition $C_1, C_2 ...., C_{|\mathcal{V}'|}$ of $\mathcal{V}$. Each cluster C_i corresponds to a super-node in $\mathcal{G}'$. The partition can also be characterized by a matrix $P' \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}'|}$, with $P'_{ij} = 1$ if and only if node v_i in $\mathcal{G}$ belongs to cluster C_i. Its normalized version can be defined by $P \triangleq P'D^{-\frac{1}{2}}$, where D is a $|\mathcal{V}| \times |\mathcal{V}'|$ diagonal matrix with $|C_i|$ as its i-th diagonal entry. The feature matrix and weighted adjacency matrix of $\mathcal{G}'$ are defined by $X' \triangleq P^T X$ and $A' \triangleq P^T AP$. After graph coarsening, the number of nodes/edges in $\mathcal{G}'$ is significantly smaller than that of $\mathcal{G}$. The coarsening rate can be defined as $c = \frac{|\mathcal{V}'|}{|\mathcal{V}|}$.

3.1.2. Adaptive Node Sampling

In a dynamic graph, the interactions of a node can be large due to the time expansion, thus making the number of neighbors also huge. By that motivation, we utilize a module Adaptive Node Sampling, which decides what neighbor will be useful for learning representation. Let h_v^t denote the node embedding vector for node v at time step t, while we denote H^t as the matrix of node embedding of all nodes at time t. Note that from initial time $t = 0$, the embedding of a node is spatial and temporal features, $H^0 = \hat{X}$.

Our goal is to learn the adaptive policy that maps the current node embedding vector - the result of Graph Transformer layers to a probability distribution over nodes. This policy guides the sampling process, determining which nodes are effectively neighbors with the target node. To optimize the policy, adversarial bandit algorithms such as Exp3 Auer et al. (2002) can be employed. These algorithms strike a balance between exploration and

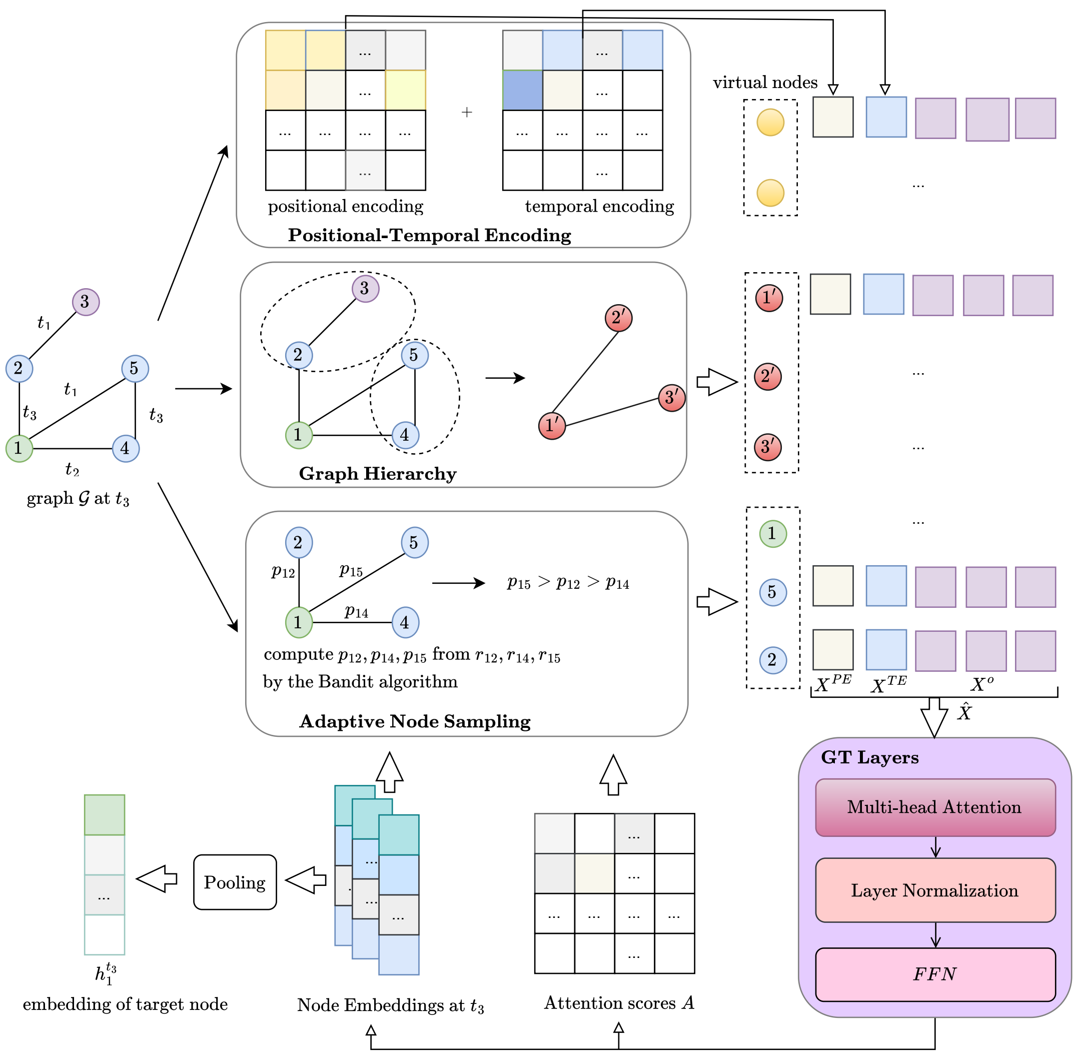

Figure 1: The ASH-DGT architecture for learning node embedding at time t_3 for the target node (index 1). The embedding of the target node (green) is updated by the sampled nodes (blue) from the Adaptive Node Sampling module and virtual nodes (red) from the Graph Hierarchy module. Extra global nodes (yellow) are added to extract the global information at the specific time step. Before sending to the Graph Transformer layers, the raw features X^0 are enhanced with the positional encoding X^{PE} and temporal encoding X^{TE}. Both the output node embedding and attention scores are then used for the sampling process at the next time step.

exploitation based on the observed rewards. The policy learning process can be formulated as follows:

$$\text{Maximize} \quad R(\mathcal{T}) = \sum_{t \in \mathcal{T}, v \in \mathcal{V}} R_v^t \tag{3}$$

$$\begin{aligned} \text{Subject to} \quad & \sum_{i \in \mathcal{N}(v)} p_i = 1, \\ & 0 < \gamma < 1 \end{aligned} \tag{4}$$

where R_v^t represents the reward obtained at time step t based on the selected subgraph of node v.

$$R_v^t = \frac{1}{N} \sum_{v \in \mathcal{S}(v)} A_v^t \cdot ||h_v^t|| \tag{5}$$

The reward function is used to evaluate the attention-based performance in our algorithm. For each node v, the contribution is computed as the product of the attention score A_v^t and the magnitude of the node embedding $||h_v^t||$. The attention score A_v^t represents the relevance or importance of node v in the attention mechanism at time step t taken from the Transformer layer as from the equation 8. It is a measure of how much attention should be given to that particular node based on its characteristics and its relationship with other nodes in the graph.

By using this reward function, we aim to quantify the performance of the attention mechanism in our algorithm solely based on the attention scores and node embeddings. The reward value R_t provides an indication of the quality and effectiveness of the attention mechanism in selecting relevant nodes at time step t. This measure can be utilized to optimize and evaluate the attention-based aspects of our algorithm. It is important to

Algorithm 1: Extent EXP3 Algorithm for node-wise sampling

```
Input   : $K$: number of chosen nodes, $\eta$: number of neighbor nodes, $\gamma$: exploration
          rate
Output: : $p$: policy distribution
Initialize $w \leftarrow (1, 1, \dots, 1)$;
for each iteration do
    Compute $p = (p_1, p_2, \dots, p_\eta)$ with
        $p_i = (1 - \gamma) \cdot \frac{w_i}{\sum_{i=1}^{\eta} w_i} + \frac{\gamma}{\eta}$
    Choose $K$ nodes based on $p$; Compute reward $r$ following eq.5,
    Update the weights using the EXP3 update:
        $w_i = w_i \cdot e^{-\eta \cdot \frac{r}{K \cdot p_i}}$
end
```

note that the reward function 5 is specifically tailored to our algorithm and the problem it addresses. Its usage allows us to assess attention-based performance and make informed decisions regarding the attention mechanism's behavior and effectiveness based on the provided attention scores and node embeddings. To adapt to the changing dynamics of the graph, the DGT model updates the node embedding vectors of the selected nodes in the subgraph. This update incorporates information from the current time step and the previous state of the graph, allowing the node embeddings to reflect the evolving nature of the graph accurately.

Algorithm 2: Adaptive Sampling Iterative Process

```
Input  : $G = (V, E, T)$: dynamic graph, $H_0$: initial node embeddings, $t$: maximum of
         time steps
Output: $H^t$: the updated node embedding
Initialize: $i \leftarrow 1$, $H \leftarrow H_0$;
while $i \leq numEpoch$ do
    for each node $v$ in $V$ do
        Train the policy $p$ following the Algorithm 1; Sample neighbor nodes from $p$;
    end
    Training DGT;
    Updated node embeddings $H^t$ for time step $t$; $i \leftarrow i + 1$
end
```

3.2. Dynamic Graph Transformer

Consider a temporal sequence S as a result of sampling by the adaptive sampler, supernodes from graph coarse, and the virtual nodes. The virtual nodes (a.k.a global nodes) help the model capture better the long-range dependencies. This sequence captures the dynamic nature of the graph, allowing us to track nodes and their interactions over time.

Positinal-Temporal Encoding For Dynamic Graph Transformer, we utilize Laplacian absolute positional encoding to incorporate spatial information and encoder time steps to embed temporal information. Formally, given a graph G with n nodes, the graph Laplacian L is computed as: $L = D - A$ where D is the degree matrix and A is the adjacency matrix of the graph. The Laplacian absolute positional encoding PE is then derived from the eigenvalues and eigenvectors of L. Let $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ be the diagonal matrix of eigenvalues and $U = [u_1, u_2, \ldots, u_n]$ be the matrix of corresponding eigenvectors. The positional encoding for node i is given by:

$$X_i^{PE} = U_i \Lambda^{-1/2} \tag{6}$$

where U_i is the i-th row of the eigenvector matrix U. Additionally, to capture temporal information, we use a learnable cosine function with weights. Each time step t is encoded using a cosine function parameterized by learnable weights W. This allows the model to learn the optimal temporal encoding during training. The temporal encoding TE_t for a time step t is defined as:

$$X_t^{TE} = \cos(W_t \cdot t + b_t) \tag{7}$$

where W_t and b_t are learnable parameters specific to the time step t, and cos denotes the cosine function. The spatial-temporal features $\hat{X}$ are transformed using multi-head self-attention mechanisms. This allows nodes to attend to other nodes within their temporal context. Note that $\hat{X} = X^{PE}||X^{TE}||X^{o}$, where X^{o} denotes original features of node. The attended embeddings are given by:

$$H^t = \text{MultiHeadAttention}(\hat{X}) \tag{8}$$

To update the embedding of node v at time t. We employ a pooling step that averages the embedding of all nodes to represent the v's interactions.

$$h_v^t = \text{MEAN}(H[c_s])W + b \tag{9}$$

where c_s is chosen nodes from node v, W, b is learning weights.

3.3. Learning in Dynamic Graphs with Downstream tasks

Node classification. During the training and inference, we sample S input sequences for each center node and use the center node representation from the final Transformer layer h_v^t for prediction. Note that the computational complexity is controllable by choosing a suitable number of sampled nodes. An MLP (Multi-Layer Perceptron) is used to predict the node class:

$$\widetilde{y}^t = f_{NC}\left(h_v^t\right) \tag{10}$$

where $\widetilde{y}^t \in R^{C\times 1}$ stands for the classification result, C is the number of classes. In the training process, we optimize the average cross entropy loss of labeled training nodes V_L :

$$\mathcal{L} = -\frac{1}{S}\sum_{v_i\in V_L}\sum_{s=1}^{S} y_i^T \log \widetilde{y}_i^t, \tag{11}$$

where $y_i \in R^{C\times 1}$ is the ground truth label of center node v_i. In the inference stage, we take a bagging aggregation to improve accuracy and reduce variance:

$$\widetilde{y}_i = \frac{1}{S}\sum_{s=1}^{S}\widetilde{y}_i^t \tag{12}$$

Link Prediction In the task of time-aware link prediction, we aim to predict the existence or absence of links between pairs of nodes in a temporal network. In both the training and inference phases in the link prediction task as (u, v), we also sample S nodes from center u and v and utilize the representations of the nodes from the final Transformer layer, denoted as h_u^t and h_v^t, respectively. To make predictions, we concatenate these representations and pass the resulting vector through a Multi-Layer Perceptron (MLP):

$$\widetilde{y}^{(s)} = f_{\text{LP}}([h_u^t], h_v^t]) \tag{13}$$

Here, $\widetilde{y}_{ij} \in R^{C\times 1}$ represents the classification result for the link between nodes v_i and v_j at time step t, where C is the number of classes. During the training process, we optimize the average cross-entropy loss for the labeled training links E_L:

$$\mathcal{L} = -\frac{1}{S}\sum_{(v_i,v_j)\in E_L}\sum_{s=1}^{S} y_{ij}^T \log \widetilde{y}_{ij}^{(s)} \tag{14}$$

Here, $y_{ij} \in R^{C\times 1}$ represents the ground truth label of the link between nodes v_i and v_j.

4. Experiments

This section represents the experimental setup for evaluating the performance of the proposed method to compare with other popular baselines.

4.1. Experimental setup

Dataset For experimental analysis, we have selected a comprehensive set of datasets that encompass a wide range of temporal dynamics across diverse domains. They include the Wikipedia temporal dataset, which captures the temporal evolution of articles; the Reddit temporal dataset Kumar et al. (2019), enabling the study of temporal patterns in online discussions; the ENRON temporal dataset Shetty and Adibi (2004), providing insights into communication dynamics within a corporate environment; the UCI temporal dataset Panzarasa et al. (2009), encompassing various domains and offering temporal information for research purposes; and the MOOC temporal dataset Kumar et al. (2019) captured temporal aspects of a online course system such as enrollment, participation, and learning behaviors over time. The detailed statistics of each dataset are described in Table 1.

We evaluate the efficiency of our model and baselines following the setting from Rossi et al. (2020). More specifically, we split the data by time for training, validating, and testing. We use the first 70% interaction to train, the next 15% to evaluate, and the final 15% to test. Note that because our proposed model can learn continuously, the duration could be changed freely.

Table 1: The detail of the datasets used in our experiments.

Dataset	Wikipedia	Reddit	SocialEvo	ENRON	UCI	MOOC
# Nodes	9227	10984	74	184	1899	7145
# Edges	157474	672447	352180	2099520	59835	411749
# Edge feature dim	172	172	0	0	0	4
# Timespan	30 days	30 days	30 days	1316 days	193days	30 days

Baselines We compare our method against a variety of strong baselines on the task of edge prediction and node classification on dynamic graphs. These baselines can be categorized into two groups. (1) Continuous-based methods: TGAT Xu et al. (2020), JODIE Kumar et al. (2019), TGN Rossi et al. (2020), and DyRep Trivedi et al. (2019). (2) Snapshot-based methods: DynAERNN Goyal et al. (2020), VGRNN Hajiramezanali et al. (2019), EvolveGCN Pareja et al. (2020), DySAT Sankar et al. (2020), and ASTGN Jiang et al. (2022).

Details setting of ASH-DGT and baselines. ASH-DGT is implemented by the PyTorch framework, and the codes for the baselines are published by the author on GitHub. We conduct experiments on a GPU RTX 3090 with all baselines and our model with 10 random seeds. The models are training with Adam Optimizer. We select the search space of hyper-parameters as described in Table 4.2. .

Evaluation Metrics. In evaluating the performance of our node classification model, we utilize the accuracy metric. Accuracy measures the proportion of correctly classified nodes and provides an overall assessment of the model's ability to classify instances correctly.

Table 2: Hyperparameter used in STASH-DGT and the baselines

Hyperparameter	Values
Learning rate	{0.01, 0.001, 0.0001, 0.0005}
Dropout	{0.1, 0.2, 0.3, 0.4, 0.5}
Batch size	{256, 512, 1024}
Number epochs	$\leq$ 1000
Attention head H	8
Number of layer L	{2, 3, 4, 5,5 6}
Number of global nodes	{1, 2, 3}
Number of super-nodes	{0, 3, 6, 9}
Number sampled neighbor nodes	{5, 10, 15, 20, 25}

In evaluating the performance of our link prediction model, we utilize multiple metrics, including AUC (Area Under the ROC Curve) and Average Precision (AP).

4.2. Results

4.2.1. Node classification task

We show the performances of our proposed ASH-DGT model alongside baseline methods for node future classification in Table 3. Our proposed model, ASH-DGT, demonstrates significantly improved performance in this task. Notably, ASH-DGT outperforms the previous best model, TGN, achieving accuracy improvements of 0.85%, 3.92%, and 1.85% on the MOOC, Wikipedia, and Reddit datasets, respectively. These results highlight the effectiveness of ASH-DGT in capturing the dynamic and hierarchical structure of graphs, leading to more accurate node classification.

Table 3: The performance of our model (Accuracy - %) and baselines on node classification task

Model	MOOC	Wikipedia	Reddit
DySAT [2019]	72.11 $\pm$ 0.5	61.79 $\pm$ 0.3	74.82 $\pm$ 1.2
Jodie [2019]	73.39 $\pm$ 2.1	61.23 $\pm$ 2.5	84.35 $\pm$ 1.2
EvolveGCN [2020]	70.26 $\pm$ 0.5	63.41 $\pm$ 0.3	81.77 $\pm$ 1.2
TGAT [2020]	74.23 $\pm$ 1.2	65.43 $\pm$ 0.7	83.12 $\pm$ 0.7
DyRep [2020]	75.12 $\pm$ 0.7	62.79 $\pm$ 2.3	84.82 $\pm$ 2.2
TGN [2020]	77.47 $\pm$ 0.8	67.11 $\pm$ 0.9	87.41 $\pm$ 0.3
ASH-DGT (ours)	79.08 $\pm$ 0.5	69.74 $\pm$ 1.3	89.12 $\pm$ 0.3

4.2.2. Link prediction task

Tables 4 and 5 present a comprehensive overview of the results attained from our proposed model ASH-DGT, juxtaposed with the performance exhibited by various baseline methods, in the domain of temporal link prediction. A thorough examination of the comparative metrics reveals a consistent trend where our model significantly outperforms the baseline

Table 4: The performance of our proposed model and baselines on the link prediction task of Wikipedia, Reddit, and MOOC datasets.

Dataset	Wikipedia		Reddit		MOOC	
Metric	AUC (%) $\uparrow$	AP (%) $\uparrow$	AUC (%) $\uparrow$	AP (%) $\uparrow$	AUC (%) $\uparrow$	AP (%) $\uparrow$
DyRep [2019]	77.40 ± 1.2	75.20 ± 1.1	67.36 ± 1.1	66.12 ± 1.0	90.49 ± 0.3	89.23 ± 0.4
JODIE [2019]	88.43 ± 0.7	87.95 ± 0.6	87.71 ± 0.4	87.43 ± 0.5	90.50 ± 0.1	90.20 ± 0.2
VGRNN [2019]	72.20 ± 0.6	71.00 ± 0.7	51.89 ± 0.2	50.56 ± 0.3	90.03 ± 0.4	89.00 ± 0.3
EvolveGCN [2020]	60.48 ± 0.5	58.74 ± 0.4	58.42 ± 0.4	57.33 ± 0.3	50.36 ± 0.8	49.76 ± 0.6
DynAERNN [2020]	71.00 ± 1.1	70.12 ± 1.0	83.37 ± 1.4	82.54 ± 1.3	89.34 ± 0.2	88.23 ± 0.3
TGAT [2020]	95.47 ± 0.1	94.89 ± 0.2	96.65 ± 0.1	95.98 ± 0.1	72.09 ± 0.3	70.95 ± 0.3
TGN-attn [2020]	95.72 ± 2.1	94.67 ± 1.9	96.12 ± 1.1	95.65 ± 1.0	81.64 ± 1.3	80.23 ± 1.1
ASTGN [2022]	96.92 ± 0.2	95.87 ± 0.3	98.97 ± 0.1	98.45 ± 0.1	92.75 ± 0.8	91.45 ± 0.7
ASH-DGT (ours)	97.45 ± 0.3	96.50 ± 0.2	99.02 ± 0.1	98.76 ± 0.1	93.23 ± 0.4	92.10 ± 0.3

Table 5: The performance of our proposed model and baselines on the link prediction task of UCI, SocialEvo, and Enron datasets.

Dataset	UCI		SocialEvo		Enron	
Metric	AP (%) $\uparrow$	AUC (%) $\uparrow$	AP (%) $\uparrow$	AUC (%) $\uparrow$	AP (%) $\uparrow$	AUC (%) $\uparrow$
DyRep [2019]	79.76 ± 0.1	81.02 ± 0.3	62.02 ± 1.7	64.35 ± 1.1	67.28 ± 1.3	68.55 ± 1.4
JODIE [2019]	78.02 ± 0.2	79.53 ± 0.5	60.01 ± 1.1	61.56 ± 1.2	63.10 ± 1.3	64.72 ± 1.0
EvolveGCN [2020]	76.63 ± 0.2	78.12 ± 0.4	56.90 ± 0.6	57.89 ± 0.5	57.37 ± 0.2	58.62 ± 0.3
TGAT [2020]	60.25 ± 0.2	61.32 ± 0.6	57.37 ± 0.6	58.91 ± 0.5	60.36 ± 0.7	61.67 ± 0.9
TGN-attn [2020]	64.21 ± 0.2	65.55 ± 0.4	58.11 ± 1.2	59.74 ± 0.8	62.47 ± 1.7	63.53 ± 1.3
ASTGN [2022]	81.52 ± 0.2	82.10 ± 0.3	62.41 ± 1.0	63.65 ± 1.2	69.12 ± 1.0	70.56 ± 0.9
ASH-DGT (ours)	82.34 ± 0.2	83.21 ± 0.2	65.12 ± 0.3	66.45 ± 0.4	71.33 ± 0.5	72.67 ± 0.4

alternatives by a considerable margin. Notably, the advancements introduced by ASH-DGT coupled with the integration of adaptive sampling manifest as a driving factor behind the observed performance boost. It is particularly intriguing to note that even in comparison to the highly adept temporal attention model ASTGN, our model showcases a notable edge, carving out a performance improvement in the range of $1 - 3\%$.

4.3. Ablation study

4.3.1. Effectiveness of graph coarsening algorithm

To comprehensively explore the influence of the Graph Hierarchy module, we conducted an experimental analysis across three datasets: Wikipedia, UCI, and EnRon. The results of this investigation are depicted in Figure 2. Notably, we observed that the choice of the number of super-nodes significantly affects the performance of our proposed method. Strikingly, for all three datasets, the AUC/AP values exhibited a prominent trend toward optimization when the number of super-nodes was set at 10. This suggests a critical point at which the trade-off between effective information utilization and computational efficiency is most favorable. While smaller numbers of super-nodes led to diminished performance due to inadequate information capture, larger numbers introduced complexities that resulted in only marginal improvements. This consistent trend across diverse datasets underscores the generalizability of the observation and points to 10 super-nodes as a strong candidate for achieving optimal representation learning outcomes.

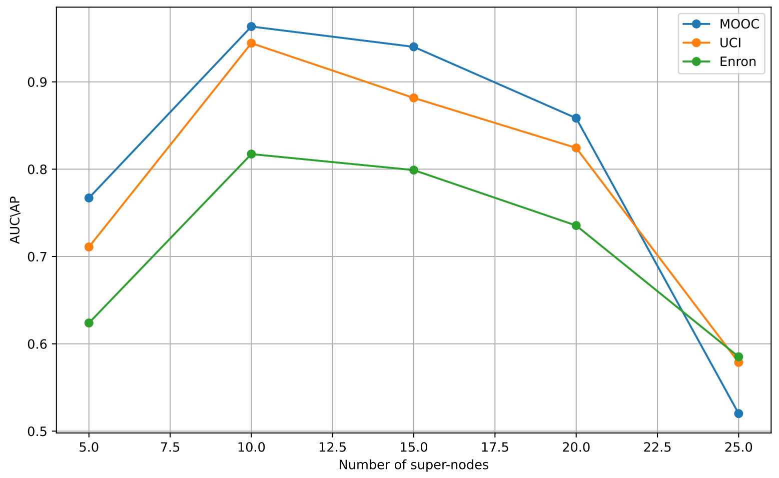

Figure 2: The performance of ASH-DGT with the number of super-nodes

In the ASH-DGT model, we use graph coarsen to compact the large graph into smaller ones, which helps the transformers easily adapt with limited computation resources. Here, we evaluate ANS-GT with different coarsening algorithms and different coarsening rates. Specifically, the considered coarsening algorithms include Variation Neighborhoods (VN) Loukas (2019), Variation Edges (VE) Loukas (2019), and Algebraic (JC) Ron et al. (2011). We vary the coarsening rate from 0.01 to 0.50. In Table 6 we show the performances of these coarsening methods on the node classification task (Wikipedia) with AUC and the link prediction task (Enron) with AP. Note that the value $c = 1.0$ denotes the use of a full graph. We can see that the graph coarsening is helpful to ASH-DGT in learning efficient representation, it can be demonstrated by the result of link prediction and node classification.

4.3.2. Effectiveness of adaptive sampling

In order to comprehensively explore the influence of the number of sampled neighbor nodes on each target node within dynamic graphs, we conducted an experimental analysis across three datasets: Wikipedia, UCI, and EnRon. The results of this investigation are depicted in Figure 2. Notably, we observed that the choice of the number of sampled neighbor nodes significantly affects the performance of our proposed method. Strikingly, for all three datasets, the AUC/AP values exhibited a prominent trend toward optimization when the number of sampled neighbor nodes was set at 10. This suggests a critical point at

Table 6: Sensitivity analysis of coarsening algorithms and coarsening rate on the node classification task (Wikipedia) and the link prediction task (Enron) without adaptive sampling.

Dataset	Method	$c = 0.01$	$c = 0.05$	$c = 0.10$	$c = 0.50$	$c = 1.00$
Wikipedia	VN	88.60	92.23	88.14	91.01	93.26
	VE	87.95	88.13	88.30	87.32	87.22
	JC	88.49	88.20	87.46	87.36	89.14
Enron	VN	70.72	69.45	70.10	68.83	69.08
	VE	69.20	69.66	67.51	68.94	68.06
	JC	69.15	69.85	69.92	70.16	69.09

which the trade-off between effective information utilization and computational efficiency is most favorable. While smaller numbers of sampled neighbor nodes led to diminished performance due to inadequate information capture, larger numbers introduced complexities that resulted in only marginal improvements. This consistent trend across diverse datasets underscores the generalizability of the observation and points to 10 sampled neighbor nodes as a strong candidate for achieving optimal representation learning outcomes.

To assess the computational efficiency of these models, we measured and compared the time required for training on four diverse datasets, the results are represented in table 7. The models are evaluated on four datasets: MOOC, UCI, Wikipedia, and Reddit. Among the models, ASH-DGT stands out as the most computationally efficient, consistently outperforming the other models across all datasets. For example, on the MOOC dataset, ASH-DGT achieves a remarkable training time of seconds per batch, significantly faster than other models such as DyRep, TGAT, and TGN-attn. Similarly, on the UCI, Wikipedia, and Reddit datasets, ASH-DGT consistently maintains its efficiency advantage. It can be seen that ASH-DGT demonstrates computational efficiency compared to the baseline models, making it a promising choice for training graph-based models, particularly in scenarios where computational resources are limited or efficiency is a critical factor.

Table 7: The comparison of the computational time for training of the proposed model and baseline. The average time is reported per batch with lower is better.

Model	MOOC	UCI	Wikipedia	Reddit
$\lvert V \rvert$	7,144	1,899	9,227	10,984
$\lvert E_{temp} \rvert$	411,749	59,835	157,474	672,447
DyRep [2019]	227.9	205.2	217.3	222.6
JODIE [2019]	182.1	178.5	187.3	191.1
TGAT [2020]	229.8	220.4	236.5	237.0
TGN-attn [2020]	242.2	225.3	260.2	250.8
ASTGN [2022]	–	158.1	190.7	212.4
ASDGT (ours)	92.1	80.2	102.4	120.3

4.3.3. Effectiveness of ASH-DGT in long period

Time Projection: Our proposed model projects the embedding to capture temporal information and predicts the future embedding at a time. After a short duration Δ_t the node i's projected embedding is updated to as follow:

$$h_{i(t+\Delta t)} = (1 + w) * h_{i(t)} \tag{15}$$

where w is time-context vector is converted from Δt by using a linear layer: $w = W_p \Delta t$. The vector $(1 + w)$ works as a temporal attention vector to scale the past node embedding.

We test the accuracy of our proposed model by varying the time projecting window Δt. The node classification task results on the Reddit dataset of our model and other baselines are shown in Table 8. In general, it is more difficult to predict for a long period updating time Δt than the short one. While all of the tested models drop accuracy, our model still achieves the best accuracies. At the larger $\Delta t = 7$, the proposed ASH-DGT achieves around 85.36% accuracy. The second highest accuracy is the TGN with 82.53% accuracy. These outcomes reinforce our model's adaptability to extended time horizons, affirming its capability to navigate the complexities of temporal dynamics and offering promising insights into its applicability across real-world scenarios.

Table 8: The accuracy of node classification task on Reddit dataset by varying the time projection $\Delta t(days)$ of different models

Model	$\Delta t = 1$	$\Delta t = 3$	$\Delta t = 5$	$\Delta t = 7$
DySAT [2019]	82.32 ± 0.7	75.13 ± 0.5	74.05 ± 0.4	71.39 ± 0.5
Jodie [2019]	84.35 ± 1.2	81.71 ± 0.8	81.13 ± 0.5	79.38 ± 0.7
EvolveGCN [2020]	81.77 ± 1.2	70.39 ± 0.7	71.22 ± 0.5	74.07 ± 0.5
DyRep [2020]	84.82 ± 2.2	80.33 ± 0.5	81.05 ± 0.5	79.77 ± 1.1
TGAT [2020]	83.12 ± 0.7	84.46 ± 0.5	83.18 ± 0.7	78.59 ± 1.2
TGN [2020]	87.41 ± 0.3	87.58 ± 0.5	86.11 ± 0.3	82.53 ± 0.5
ASH-DGT (ours)	89.28 ± 0.3	88.15 ± 0.2	86.62 ± 0.5	85.36 ± 0.6

5. Conclusion

In this work, we investigate adaptive sampling strategies for enhancing representation learning within the context of dynamic graphs using graph transformers. Our investigation has shed light on the critical role of adaptively selecting nodes during training to capture both short- and long-range dependencies, thereby improving the model's efficacy in capturing dynamic interactions. Through comprehensive analysis and experimentation, we've showcased the potential of our proposed adaptive sampling technique to yield significant enhancements in performance, resulting in improved accuracy and convergence. This research marks a substantial stride forward in the field of representation learning, addressing the challenges posed by dynamic graph data and offering new avenues for refining graph-based models.

Acknowledgements

This research was funded by the research project QG.23.32 of Vietnam National University, Hanoi. This project is also supported by QUB Agility+ funding, DfE project (R3648ECS) and UKRI/EU funded project RELAX (R1142ECS).

References

Nahla Mohamed Ahmed, Ling Chen, Yulong Wang, Bin Li, Yun Li, and Wei Liu. Sampling-based algorithm for link prediction in temporal networks. *Information Sciences*, 374:1–14, 2016.

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002.

Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowl.-Based Syst.*, 151:78–94, 2018.

Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowl.-Based Syst.*, 187:104816, 2020.

Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. *NeurIPS*, 32, 2019.

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

Thi Linh Hoang, Tuan Dung Pham, and Viet Cuong Ta. Improving graph convolutional networks with transformer layer in social-based items recommendation. In *KSE*, pages 1–6. IEEE, 2021.

EunJeong Hwang, Veronika Thost, Shib Sankar Dasgupta, and Tengfei Ma. Revisiting virtual nodes in graph neural networks for link prediction. *LoG*, 2022.

Wenjian Jiang, Xuhong Wang, Ping Cui, Bin Huang, Yupu Yang, and Qifu Fan. Adaptive sampling temporal graph network. In *CTISC*, pages 1–8, 2022.

Weiyang Kong, Ziyu Guo, and Yubao Liu. Spatio-temporal pivotal graph neural networks for traffic flow forecasting. In *AAAI*, volume 38, pages 8627–8635, 2024.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*, pages 1269–1278, 2019.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.*, 20(116):1–42, 2019.

Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *J. Assoc. Inf. Sci. Technol.*, 60(5):911–932, 2009.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*, volume 34, pages 5363–5370, 2020.

Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Model. Simul.*, 9(1):407–423, 2011.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM*, pages 519–527, 2020.

Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. *Information sciences institute technical report, University of Southern California*, 4(1):120–128, 2004.

Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9: 79143–79168, 2021.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *ICLR*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *NeurIPS*, 31, 2018.

Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. *NeurIPS*, 36:67686–67700, 2023.