

Dynamic Graph Node Classification via Time Augmentation

Jiarui Sun^{*}, Mengting Gu[†], Chin-Chia Michael Yeh[†], Yujie Fan[†], Girish Chowdhary[‡], Wei Zhang[†]

^{*}Dept. of Electrical and Computer Engineering,

University of Illinois at Urbana, Champaign, Urbana, USA

[‡]Dept. of Computer Science and Dept. of Agricultural and Biological Engineering,

University of Illinois at Urbana, Champaign, Urbana, USA

[†]Visa Research, Palo Alto, USA

{jsun57,girishc}@illinois.edu, {mengu,miyeh,yufan,wzhan}@visa.com

Abstract—Node classification for graph-structured data aims to classify nodes whose labels are unknown. While studies on static graphs are prevalent, few studies have focused on dynamic graph node classification. Node classification on dynamic graphs is challenging for two reasons. First, the model needs to capture both structural and temporal information, particularly on dynamic graphs with a long history and require large receptive fields. Second, model scalability becomes a significant concern as the size of the dynamic graph increases. To address these problems, we propose the Time Augmented Dynamic Graph Neural Network (*TADGNN*) framework. *TADGNN* consists of two modules: 1) a time augmentation module that captures the temporal evolution of nodes across time structurally, creating a time-augmented spatio-temporal graph, and 2) an information propagation module that learns the dynamic representations for each node across time using the constructed time-augmented graph. We perform node classification experiments on four dynamic graph benchmarks. Experimental results demonstrate that *TADGNN* framework outperforms several static and dynamic state-of-the-art (SOTA) GNN models while demonstrating superior scalability. We also conduct theoretical and empirical analyses to validate the efficiency of the proposed method. Our code is available *here*.

Index Terms—graph neural network, node classification, dynamic graph

I. INTRODUCTION

Graph is a ubiquitous data structure that represents relationships between entities. Aiming to classify graph nodes whose labels are unknown, the task of node classification for graph-structured data has recently received increasing attention in various domains, such as biology [1] and social sciences [2]. This is attainable by the recent advance of Graph Neural Networks (GNNs), which generate node representations for classification through a message passing mechanism where each node iteratively aggregates neighborhood information.

Existing GNNs mainly focus on static graphs [3], [4]. However, many real-world graphs are dynamic. Dynamic graphs can generally be categorized into discrete-time dynamic graphs and continuous-time dynamic graphs according to their representations. Discrete-time dynamic graphs use an ordered sequence of graph snapshots, where each snapshot represents aggregated dynamic information within a fixed time interval.

Continuous-time dynamic graphs maintain detailed temporal information and are often more complex to model than the discrete case. For both cases, graph structure, node attributes and node labels evolve over time in a complex manner. Performing node classification on such dynamic graphs demands capturing this complicated evolving nature, which not only requires exploring the structural topology but also modeling the temporal aspect, which is absent in static GNN models.

In this work, we focus on node classification on discrete-time dynamic graphs. Because the existing dynamic graph learning methods are inefficient in both time and space as they either rely on recurrent structures [5] or the attention mechanism [6], these methods are not applicable for domains where dynamic graphs with a large number of time steps and nodes inhabit. To this end, we propose a novel GNN framework named *Time Augmented Dynamic Graph Neural Network (TADGNN)*. *TADGNN* first employs a time augmentation module that constructs a time-augmented spatio-temporal graph based on the original graph snapshots. The time augmentation module *efficiently* realizes the temporal evolution of nodes across time in a structural sense. An information propagation module is then applied to *effectively* learn the spatio-temporal information by aggregating node features from time-augmented neighborhoods. Comparing with existing literature, *TADGNN* achieves a good efficiency in two folds. First, the time augmentation module models temporal graph dynamics without introducing significant computational demand since the time-augmented graph construction does not require learning. In addition, by learning different importance scores for time-augmented neighborhoods of each node once, the dynamic attention mechanism introduced in the information propagation module enhances the model capacity while introducing little to no computational overhead. These advantages make *TADGNN* both powerful and efficient. We summarize our key contributions as below:

- A novel GNN framework named *TADGNN* that works efficiently for node classification problems on dynamic graphs.
- A theoretical and empirical analysis that validates the training efficiency of *TADGNN*.
- A comprehensive set of experiments that demonstrate the

^{*}Work done while at Visa Research.

effectiveness of *TADGNN* over SOTA methods.

II. RELATED WORK

A. Node Classification on Static Graphs

Many works address static graph node classification problem by exploiting random walk statistics to optimize a stochastic measure of node similarities [7], [8]. Another line of research relies on graph spectral properties, designing convolutional filters based on the spectral graph theory [9], [10]. One significant work in this domain is Graph Convolutional Network (GCN) [3], which restricts spectral filters to operate on each node's immediate neighbors, and can be interpreted as spatial aggregation. It also inspires more spatial-based graph convolutional neural network models [4], [11], [12].

B. Node Classification on Dynamic Graphs

1) *Discrete-time Dynamic Graphs*: Many existing works utilize recurrent models for discrete-time dynamic graphs to capture the temporal dynamics into hidden states for classification. Some works use separate GNNs to model individual graph snapshot and use RNNs to learn temporal dynamics [5]; some other works integrate GNNs and RNNs together into one layer, aiming to learn the spatial and temporal information concurrently [13]. Sankar et al. [6] use the self-attention mechanism along both the spatial and temporal dimensions of dynamic graphs, while Xu et al. [1] combine the self-attention mechanism and RNNs together to learn the spatio-temporal contextual information jointly.

2) *Continuous-time Dynamic Graphs*: Existing works on continuous-time dynamic graphs include RNN-based methods, temporal walk-based methods and temporal point process-based methods. RNN-based methods perform node updates through recurrent models at fine-grained timestamps [14], and the other two categories incorporate temporal dependencies through temporal random walks and parameterized temporal point processes [15], [16].

III. PRELIMINARIES

In this work, we focus on classifying nodes for discrete-time dynamic graphs $\mathbb{G} = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^T\}$. Each snapshot is an undirected graph $\mathcal{G}^t = (\mathcal{V}, \mathcal{E}^t, \mathcal{X}^t)$ at time step t . \mathcal{V} denotes the set of all the nodes appeared in \mathbb{G} , and $|\mathcal{V}| = N$ is the number of all nodes. $\mathcal{E}^t \subseteq \mathcal{V} \times \mathcal{V}$ denotes the edge set of the snapshot at time t . The edge set can also be represented as an adjacency matrix $A^t \in \mathbb{R}^{N \times N}$ where $A_{uv}^t = 1$ if $(u, v) \in \mathcal{E}^t$ otherwise $A_{uv}^t = 0$. $\mathcal{X}^t \in \mathbb{R}^{N \times d}$ denotes the node attribute matrix at time step t where d is the initial node feature dimension, and $\mathbf{x}_v^t \in \mathbb{R}^d$ is the feature vector of node v . \mathcal{Y}^t denotes the class label associated with each node at time step t , and \mathbf{y}_v^t is the class label of node v . All nodes have their dynamic labels which evolve over time.

Problem Statement: Given the dynamic graph history $\mathbb{G}_L = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^L\}$ and the corresponding node labels $\mathbb{Y}_L = \{\mathcal{Y}^1, \mathcal{Y}^2, \dots, \mathcal{Y}^L\}$ up to time step L , we aim to classify all the nodes in future snapshots $\mathbb{G}_U = \{\mathcal{G}^{L+1}, \mathcal{G}^{L+2}, \dots, \mathcal{G}^T\}$ whose node labels $\mathbb{Y}_U = \{\mathcal{Y}^{L+1}, \mathcal{Y}^{L+2}, \dots, \mathcal{Y}^T\}$ are unknown.

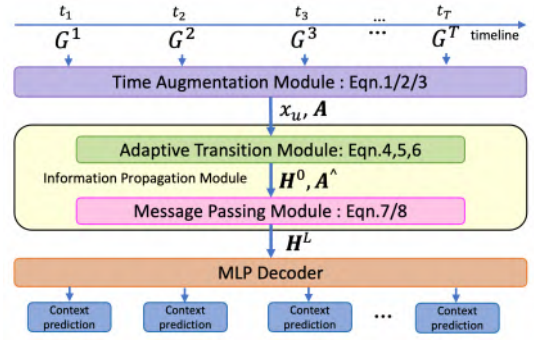


Fig. 1. *TADGNN* model architecture.

IV. TADGNN ARCHITECTURE

In this section, we present our framework, namely *Time Augmented Dynamic Graph Neural Network (TADGNN)*, which is depicted in Fig. 1. We first introduce the proposed time augmentation module. Then, we describe the information propagation module. Finally, we discuss the time complexity of *TADGNN* and compare it with several SOTA methods.

A. Time Augmentation Module

The time augmentation module is designed to model the temporal evolution of nodes across different snapshots in a structural sense. To help explain our method, we define the *temporal walk* under *discrete-time dynamic graph* setting.

Definition IV.1 (Temporal Walk). Given a discrete-time dynamic graph sequence $\mathbb{G} = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^T\}$, a *temporal walk* from v_1 to v_{k+1} is defined as $((v_1, v_2, t_1), (v_2, v_3, t_2), \dots, (v_k, v_{k+1}, t_k))$ such that $(v_i, v_{i+1}) \in \mathcal{E}^{t_i}$, $\mathcal{G}^{t_i} = (\mathcal{V}, \mathcal{E}^{t_i})$ for $1 \leq i \leq k$, and $1 \leq t_1 \leq t_2 \leq \dots \leq t_k \leq T$.

A temporal walk is a time-respected walk that represents natural information flow through dynamic graph, but it is rarely explored in the literature. To model such time-respected behaviors, we design the time augmentation module, which is inspired by the temporal PageRank algorithm [17]. The time augmentation module aims to construct a time-augmented spatio-temporal graph, the walks on which can be considered as temporal walks simulated on the original dynamic graph. In this work, we consider three different realizations of such a time-augmented graph: the *full time-augmentation realization*, the *self-evolution time-augmentation realization*, and the *disentangled time-augmentation realization*. In the first two cases, the information propagation module is directly applied on the constructed time-augmented graph, propagating information structurally and temporally in a joint manner. In the last case, the time-augmented graph is disentangled into a structural graph and a temporal graph, where different information propagation modules are applied on these two separate graphs as a two-stage process.

1) *Full Time-Augmentation Realization*: The first realization aims to produce a bijection between all possible walks on the time-augmented graph and all possible temporal walks

on the original temporal graph. Formally, we denote the adjacency matrix of the constructed time-augmented graph as $\mathbf{A} \in \mathbb{R}^{TN \times TN}$. Then, the adjacency matrix of the time-augmented graph can be represented as:

$$\mathbf{A} = \begin{bmatrix} \tilde{\mathbf{A}}^1 & \mathbf{A}^2 & \mathbf{A}^3 & \dots \\ 0 & \tilde{\mathbf{A}}^2 & \mathbf{A}^3 & \dots \\ 0 & 0 & \tilde{\mathbf{A}}^3 & \dots \\ \vdots & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & \tilde{\mathbf{A}}^T \end{bmatrix}, \quad (1)$$

where $\tilde{\mathbf{A}}^t \in \mathbb{R}^{N \times N}$ is the corresponding adjacency matrix at time step t with self-loops added.

2) *Self-Evolution Time-Augmentation Realization*: In the self-evolution time-augmentation realization, we restrict the cross-time walks such that only temporally adjacent nodes can be attended by their historical versions. Formally, let v^t and v^{t+1} denote node v at time t and $t+1$. We only construct edge (v^t, v^{t+1}) . By doing so, all temporal walks can be realized by performing random walk on the time-augmented graph as it can visit a node's neighbor at different time steps by visiting itself at different time steps first. With previous notation, the adjacency matrix of the time-augmented graph can be represented as:

$$\mathbf{A} = \begin{bmatrix} \tilde{\mathbf{A}}^1 & \mathbf{I} & 0 & \dots \\ 0 & \tilde{\mathbf{A}}^2 & \mathbf{I} & \dots \\ 0 & 0 & \tilde{\mathbf{A}}^3 & \dots \\ \vdots & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & \tilde{\mathbf{A}}^T \end{bmatrix}, \quad (2)$$

where $\mathbf{I} \in \mathbb{R}^{N \times N}$ is the identity matrix modeling node version updates across time. It is important to note that this realization of the time-augmented graph is relatively simplified and imposes trivial overhead in model space complexity.

3) *Disentangled Time-Augmentation Realization*: In the third case where we aim to completely disentangle structural and temporal modeling, we decouple the self-evolution time-augmentation case. Instead of considering one time-augmented graph \mathbf{A} , we decouple it into $\mathbf{A}_s \in \mathbb{R}^{TN \times TN}$ and $\mathbf{A}_t \in \mathbb{R}^{TN \times TN}$ as below:

$$\mathbf{A}_s = \begin{bmatrix} \tilde{\mathbf{A}}^1 & 0 & 0 & \dots \\ 0 & \tilde{\mathbf{A}}^2 & 0 & \dots \\ 0 & 0 & \tilde{\mathbf{A}}^3 & \dots \\ \vdots & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & \tilde{\mathbf{A}}^T \end{bmatrix}, \mathbf{A}_t = \begin{bmatrix} \mathbf{I} & \mathbf{I} & 0 & \dots \\ 0 & \mathbf{I} & \mathbf{I} & \dots \\ 0 & 0 & \mathbf{I} & \dots \\ \vdots & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & \mathbf{I} \end{bmatrix}. \quad (3)$$

B. Information Propagation Module

The information propagation module aims to capture both the structural and temporal properties of the dynamic graph via aggregating information from each node's time-augmented neighborhoods. We use \mathcal{V}_{ta} , \mathcal{E}_{ta} to denote the node set and edge set of the time-augmented graph respectively, and $|\mathcal{V}_{ta}| = N \times T$. Thus, the input to the information propagation module

is the initial node representations: $\{\mathbf{x}_v^t \in \mathbb{R}^d, \forall v^t \in \mathcal{V}_{ta}\}$. For convenience, we drop the superscript on node feature vectors and nodes as \mathbf{x}_v and v .

1) *Adaptive Transition Module*: First, inspired by [18], we employ the dynamic attention mechanism to compute the adaptive information transition matrix, which allows different importance of nodes to be learned. Formally, the adaptive graph transition matrix $\hat{\mathbf{A}} \in \mathbb{R}^{TN \times TN}$ is computed as:

$$e_{uv} = \mathbf{a}^T \sigma_{att}(\mathbf{A}_{uv} \cdot ([\Theta_L, \Theta_R][\mathbf{x}_u || \mathbf{x}_v])), \quad (4)$$

$$\alpha_{uv} = \frac{\exp(e_{uv})}{\sum_{u \in \mathcal{N}_v} \exp(e_{uv})}, \quad (5)$$

$$\hat{\mathbf{A}}_{uv}^T = \alpha_{uv}, \quad (6)$$

where $\Theta_L \in \mathbb{R}^{F \times d}$ and $\Theta_R \in \mathbb{R}^{F \times d}$ are the adaptive weight matrices applied on initial node features, $[\cdot, \cdot]$ is the horizontal concatenation of two matrix, $[\cdot || \cdot]$ is the vector concatenation operation, $\mathbf{a} \in \mathbb{R}^F$ is a learnable weight vector for attention calculation, $\sigma_{att}(\cdot)$ is the LeakyReLU activation, $\mathcal{N}_v = \{u \in \mathcal{V}_{ta} : (u, v) \in \mathcal{E}_{ta}\}$ denotes the immediate neighbor set of node v on the time-augmented graph, and α_{uv} denotes the normalized attention coefficient of link (u, v) . For the disentangled time-augmentation case, the adaptive structural and temporal transition matrices $\hat{\mathbf{A}}_s \in \mathbb{R}^{TN \times TN}$ and $\hat{\mathbf{A}}_t \in \mathbb{R}^{TN \times TN}$ are computed separately following the exact same procedures but with different parameters. We would like to emphasize that the dynamic attention mechanism we introduced does not impose quadratic computational complexity with respect to the number of snapshots as previously mentioned attention-based models [6] since the attention scores are computed structurally for time-augmented neighborhoods.

2) *Message Passing Module*: In order to capture long range temporal and structural dependencies on the time-augmented graph, deep architectures are necessary. However, it is well-known that some GNN models such as GCN [3] and GAT [4] demonstrate significant performance degradation when the number of layers increases [19]. Thus, we adapt the message passing mechanism from GCNII [20] which relieves such over-smoothing issue through initial residual and identity mapping techniques. Formally, we denote the number of stacked information propagation layers as L and the node representation output from the l^{th} layer for the time-augmented graph as $\{\mathbf{h}_v^l \in \mathbb{R}^F, \forall v \in \mathcal{V}\}$ where F is the node representation dimension. The input is either the initial encoded or intermediate node representations: $\{\mathbf{h}_v^{l-1} \in \mathbb{R}^F, \forall v \in \mathcal{V}\}$. When $l = 1$, i.e., at the first layer, we have $\mathbf{h}_v^0 = \Theta_R \mathbf{x}_v$. Compactly, we have $\mathbf{H}^l \in \mathbb{R}^{TN \times F}$. Then, we can define the message passing mechanism as:

$$\mathbf{H}^{l+1} = \sigma_{ip} \left(((1 - \alpha_l) \hat{\mathbf{A}} \mathbf{H}^l + \alpha_l \mathbf{H}^0) \right. \\ \left. ((1 - \beta_l) \mathbf{I} + \beta_l \mathbf{W}^l) \right), \quad (7)$$

where $\mathbf{W}^l \in \mathbb{R}^{F \times F}$ is the weight matrix, $\mathbf{I} \in \mathbb{R}^{F \times F}$ is the identity matrix, σ_{ip} is the ReLU activation, α_l and β_l are two hyperparameters adopting the same practice of

TABLE I
SPACE AND TIME COMPLEXITY COMPARISON BETWEEN *TADGNN* AND OTHER BASELINES.

Model Type	Space Complexity	Time Complexity	Sequential Ops.
<i>TADGNN</i>	$O(ET + LF^2 + LTNF)$	$O(LETF + LTNF^2)$	$O(1)$
DySAT	$O(LET + LF^2 + LTNF + LNT^2)$	$O(LETF + LTNF^2 + LNT^2F)$	$O(1)$
EvolveGCN	$O(ET + LTF^2 + LTNF)$	$O(LETF + LTNF^2)$	$O(T)$

GCNII [20]. We also adapt GCNII variant, whose message passing mechanism is defined as:

$$\mathbf{H}^{l+1} = \sigma_{ip} \left((1 - \alpha_l) \hat{\mathbf{A}} \mathbf{H}^l ((1 - \beta_l) \mathbf{I} + \beta_l \mathbf{W}_1^l) + \alpha_l \mathbf{H}^0 ((1 - \beta_l) \mathbf{I} + \beta_l \mathbf{W}_2^l) \right), \quad (8)$$

where different weights $\mathbf{W}_1^l, \mathbf{W}_2^l$ are used for aggregated node representations $\hat{\mathbf{A}} \mathbf{H}^l$ and initial residual representations \mathbf{H}^0 . Again, for the disentangled time-augmentation case, we compute the node representations \mathbf{H} by first utilizing structural transition matrix $\hat{\mathbf{A}}_s$ to capture structural properties and then using temporal transition matrix $\hat{\mathbf{A}}_t$ to learn temporal dynamics, with different weight matrices \mathbf{W}_s^l and \mathbf{W}_t^l . With $\mathbf{A}_s, \mathbf{A}_t$ as the structural and temporal graph respectively, we first use one information propagation module for the structural graph \mathbf{A}_s . After we obtain \mathbf{H}_s which summarizes structural information, we apply the second information propagation module guided by the temporal graph \mathbf{A}_t , with \mathbf{H}_s as the initial node embedding input. Finally, we use \mathbf{H}^L that summarizes both structural and temporal dynamics for the dynamic node classification task.

C. Learning Algorithm

The outputs of the information propagation module \mathbf{H}^L are then feed into a multilayer perceptron (MLP) decoder which converts node representations to class logits, and are optimized to classify nodes in \mathbb{G}_L correctly as following:

$$\mathcal{L} = \sum_{v \in \mathbb{G}_L} J(y_v, \hat{y}_v), \hat{y}_v = \text{MLP}(\mathbf{h}_v^L), \quad (9)$$

where $J(\cdot)$ measures weighted cross-entropy loss between groundtruth y_v and predicted score \hat{y}_v . The loss weights are calculated based on class distribution from training partition.

D. Complexity Analysis

In this section, we compare *TADGNN*'s space and time complexity with DySAT and EvolveGCN, which can be considered as the representatives of attention-based and RNN-based dynamic graph learning models. The space and time complexity for each method is shown in Table I. From the table, we can observe that comparing with both DySAT and EvolveGCN, *TADGNN* has the lowest space complexity since DySAT is dominated by $O(LNT^2)$ term and EvolveGCN is dominated by $O(LTF^2)$ term. In practice, memory space is a limiting factor for DySAT and EvolveGCN when N and L are necessarily large. From the temporal perspective, the overall time complexity of *TADGNN* is the lowest among the select baselines. DySAT's time complexity includes a T^2 term that

TABLE II
DATASET STATISTICS

Datasets	Nodes	Edges	Timestamps	Classes	Split
WIKI	9,227	2,833	11	2	4/3/4
REDDIT	10,984	18,928	11	2	4/3/4
ML-RATING	9,746	90,928	11	5	4/3/4
ML-GENRE	9,704	90,334	11	15	4/3/4

makes it inefficient when modeling dynamic graphs with a large T . As an RNN-incorporated GNN model, EvolveGCN has sequential operation dependence, which makes it infeasible to be processed in parallel and makes its practical training time significantly slower than purely convolution-based methods.

V. EXPERIMENTS

In this section, we evaluate the effectiveness of *TADGNN* for dynamic node classification task on four real-world datasets by comparing with four SOTA baselines.

A. Datasets

We use four real-world dynamic graph datasets to conduct experiments, including two social networks, WIKI [21] and REDDIT [22] and two rating networks, ML-RATING [23] and ML-GENRE [23]. All datasets are sliced into graph snapshot sequences. Each snapshot contains information during fixed time intervals based on the timestamps provided in the raw data. We also make sure that each snapshot contains sufficient interactions/links between nodes. Note that though these datasets are not attributed due to the difficulty of acquiring public attributed dynamic graphs, *TADGNN* is designed for attributed dynamic graphs. As such, we use one-hot encoding of node IDs as node features in our experiments. Detailed dataset statistics are summarized in Table II.

B. Experimental Setup

We select four SOTA node classification algorithms, including GAT [4], GCNII [20], EvolveGCN [13] and DySAT [6] to conduct model evaluation. We use PyTorch to implement *TADGNN* along with all other baselines. We set the output feature dimension of the information propagation module to 128. Adam [24] is used as the optimizer with learning rate of 0.01 along with weight decay of 0.0005 as regularization to train all models for 200 epochs in all experiments. In particular, for *TADGNN*, we use validation set performance to tune the hyper-parameters and select the best time augmentation module type from three realizations. For baselines, we select hyper-parameters following the papers'

TABLE III
DYNAMIC NODE CLASSIFICATION MACRO-AUC RESULTS WITH STD.

Model	WIKI	REDDIT	ML-RATING	ML-GENRE
GAT	63.1 \pm 2.8	62.9 \pm 1.0	67.8 \pm 3.0	81.2 \pm 2.6
GCNII	67.5 \pm 1.6	61.1 \pm 1.4	71.5 \pm 3.7	62.1 \pm 1.2
DySAT	54.3 \pm 4.3	66.5 \pm 0.3	53.9 \pm 5.8	51.9 \pm 1.8
EvolveGCN	58.9 \pm 2.1	62.2 \pm 1.6	61.3 \pm 2.5	51.1 \pm 2.5
TADGNN	69.2 \pm 1.8	66.7 \pm 1.6	73.7 \pm 0.2	75.0 \pm 1.5

tuning guidelines. We report the averaged results along with corresponding standard deviations of five runs. All models share the same MLP decoder architecture with dropout ratio 0.3 to covert dynamic node representations to class logits.

C. Node Classification Experiments

In this section, we describe the conducted experiments and report the results together with the observed insights.

1) *Task Description*: Dynamic node classification task is used to evaluate *TADGNN*'s effectiveness compared with other baselines. All models are trained based on the training partition $\mathbb{G}_L = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^t\}$ with label information. The task is to predict node labels in $\mathbb{G}_U = \{\mathcal{G}^{t+1}, \mathcal{G}^{t+2}, \dots, \mathcal{G}^T\}$ by using the trained model with the entire dynamic graph $\mathbb{G} = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^T\}$ as input. Note that *TADGNN* is inductive since it is agnostic to number of input graph snapshots.

2) *Experiment Setting*: Each dataset is sliced into a discrete graph snapshot sequence where each snapshot corresponds to a fixed time interval that contains a sufficient number of links. In each set of experiments, the first t snapshots are used for model training. After training, we use the next t' snapshots for validation, and the rest $T - t - t'$ snapshots for testing. For all train/validation/test stages, *TADGNN* processes input graph snapshots all at once. We show the train/validation/test split statistics in Table II. Weighted cross-entropy loss is used as the objective function, that class weights obtained from training partition are used to relieve data imbalance issue. The original DySAT model is temporally transductive as it cannot easily incorporate new graph snapshots for testing. Thus, we modify its position embeddings [25] to sine and cosine positional encodings as in Transformer [26]. For the static baselines, we employ one shared model across all snapshots. All models are trained end-to-end.

3) *Evaluation Metric*: Since label information for all four datasets are highly imbalanced, we select *Area Under the Receiver Operating Characteristic Curve* (AUC) metric to measure performance of different models. We use macro-AUC scores for evaluation. Macro-AUC is computed by treating performances from all classes across all time steps (i.e., snapshots) equally, which is desirable for the case of dynamic node classification.

4) *Results and Discussion*: We show the macro-AUC results in Table III. Our observations include:

- *TADGNN* achieves superior performance on most of the datasets. This indicates that *TADGNN* can better capture both structural and temporal graph dynamics comparing to

TABLE IV
ABLATION STUDY OF *TADGNN*.

Components			Perf.	
Time	Augmentation	Adaptive Transition	REDDIT	ML-RATING
	\times	\times	61.4 \pm 2.8	71.2 \pm 0.6
	\checkmark	\times	63.8 \pm 0.8	70.9 \pm 1.3
	\checkmark	\checkmark	66.7 \pm 1.6	73.7 \pm 0.2

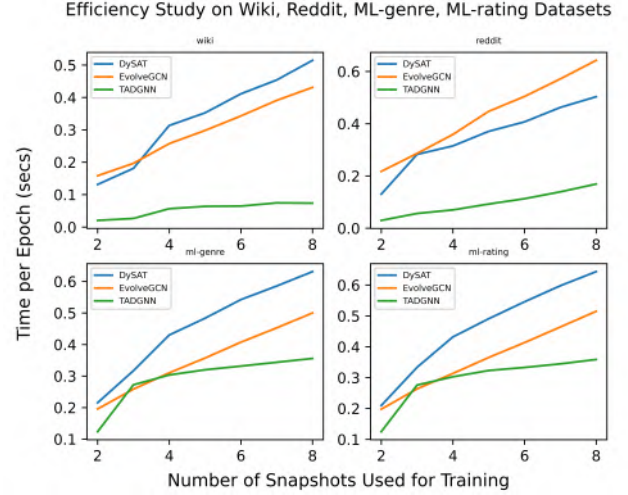


Fig. 2. Efficiency comparison among *TADGNN*, DySAT, and EvolveGCN.

other methods. On one dataset (ML-GENRE) GAT performed well and has better result, but its performances have larger variance on different datasets. In addition, *TADGNN* is more efficient in space and time complexity, and can be applied in more scenarios especially where the dynamic graphs are large.

- Other dynamic baselines have inferior performance on certain datasets comparing to static methods. Besides, these dynamic baselines tend to keep larger variances for most datasets, which suggests that *TADGNN* is more robust to random weight initialization. The results from our hyper-parameter search and analysis further indicate that the performance of these methods can be sensitive to hyper-parameter values; however, large time cost and space cost limit their potentials. This further suggests the importance of using temporal information efficiently to guide the dynamic graph node classification problem.

D. Efficiency Comparison

In this section, we empirically demonstrate the efficiency advantage of *TADGNN*. Specifically, we compare our model to EvolveGCN and DySAT on the average training time per epoch using different number of training snapshots. These two methods are chosen since they represent widely used RNN-based and attention-based dynamic graph learning models. For all three models, while keeping all common settings (i.e., learning rate) the same, we employ the best performing hyper-parameter setups. In particular, we compute the training time

used per epoch averaged across 200 epochs from 2 to 8 snapshots as training partition across all four datasets.

The efficiency comparison is shown in Fig. 2. The results are expected, as the training time of *TADGNN* demonstrates a small increasing rate with respect to the number of training snapshots, while both DySAT and EvolveGCN depict much larger increasing rate as the number of training snapshots increases, due to the self-attention mechanism and sequential operation in RNN respectively. This empirical result confirms the efficiency advantage of *TADGNN* over other deep dynamic graph learning methods. More importantly, as the number of training snapshots and number of layers increase, both DySAT and EvolveGCN quickly fill up most of the GPU memories, thus hardly scalable to longer sequences or multi-layer setups, due to the extra memory requirements as discussed in Section IV-D. In contrast, *TADGNN* takes much less memory even when much more layers inhabit, which allows *TADGNN* to capture long range temporal and structural properties when necessary. This empirical result validates our theoretical complexity analysis, demonstrating better efficiency of *TADGNN*, that it is powerful in modeling large dynamic graph datasets.

E. Ablation Study

We conduct an ablation study to investigate how the time augmentation and information propagation modules affect *TADGNN*'s modeling ability. Specifically, we prepare three *TADGNN* variants, i.e., 1) disable both the time augmentation and adaptive transition modules; 2) only disable the adaptive transition module; and 3) the full *TADGNN* setup, and observe how disabling of different components affect the model performance. We select two datasets (REDDIT and ML-RATING) to cover different types of dynamic graphs. We summarize our observations as below:

- Both the time augmentation and adaptive transition modules are vital in temporal dynamics modeling, as they help *TADGNN* achieve the best node classification results among different setups. In particular, we observe that our designed modules significantly boost *TADGNN* performance on REDDIT. The results suggest that temporal information is imperative to address dynamic node classification task.
- Coupling the time augmentation and adaptive transition modules together helps *TADGNN* reduce performance variance. We observe that enabling the time augmentation module only for ML-RATING increases performance variance significantly; on the contrary, though enabling time augmentation module only for REDDIT reduces performance variance to the minimum, the performance can be further improved by introducing the adaptive learning module. This indicates the importance of distinguishing different levels of importance of node's time-augmented neighborhoods.

VI. CONCLUSION

Node classification on dynamic graphs has been gaining considerable attention. In this paper, we propose the Time Augmented Dynamic Graph Neural Network (*TADGNN*)

framework for discrete-time dynamic graph node classification. *TADGNN* consists of two modules: 1) a time augmentation module for converting the dynamic graph to a time-augmented spatio-temporal graph and 2) an information propagation module to process the time-augmented spatio-temporal graph. Evaluations are performed on four real-world dynamic graph benchmarks. Experimental results illustrate that *TADGNN* outperforms SOTA GNN models in terms of both accuracy and efficiency.

REFERENCES

- [1] D. Xu, W. Cheng, D. Luo, X. Liu, and X. Zhang, "Spatio-temporal attentive RNN for node classification in temporal attributed graphs," in *IJCAI*, 2019, pp. 3947–3953.
- [2] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *CoRR*, vol. abs/2006.10637, 2020.
- [3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
- [4] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.
- [5] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, 2020.
- [6] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *WSDM*. ACM, 2020, pp. 519–527.
- [7] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*. ACM, 2016, pp. 855–864.
- [8] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*. ACM, 2014, pp. 701–710.
- [9] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, 2019.
- [10] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Graph neural networks with convolutional ARMA filters," *CoRR*, vol. abs/1901.01343, 2019.
- [11] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
- [12] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.
- [13] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegc: Evolving graph convolutional networks for dynamic graphs," in *AAAI*, 2020.
- [14] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *KDD*. ACM, 2019.
- [15] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *WWW*. ACM, 2018, pp. 969–976.
- [16] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *ICLR*, 2019.
- [17] P. Rozenshtein and A. Gionis, "Temporal pagerank," in *ECML PKDD*, vol. 9852. Springer, 2016, pp. 674–689.
- [18] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" *CoRR*, vol. abs/2105.14491, 2021.
- [19] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018, pp. 3538–3545.
- [20] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *ICML*, 2020.
- [21] "Wikipedia edit history dump," https://meta.wikimedia.org/wiki/Data_dumps.
- [22] "Reddit data dump," <http://files.pushshift.io/reddit/>.
- [23] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, dec 2015.
- [24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
- [25] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *ICML*, vol. 70. PMLR, 2017, pp. 1243–1252.
- [26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.