# Decoupled Graph Neural Networks for Large Dynamic Graphs

Yanping Zheng
Renmin University of China
Beijing, China
zhengyanping@ruc.edu.cn

Zhewei Wei*
Renmin University of China
Beijing, China
zhewei@ruc.edu.cn

Jiajun Liu
Data 61, CSIRO
Pullenvale, Queensland, Australia
jiajun.liu@csiro.au

## ABSTRACT

Real-world graphs, such as social networks, financial transactions, and recommendation systems, often demonstrate dynamic behavior. This phenomenon, known as graph stream, involves the dynamic changes of nodes and the emergence and disappearance of edges. To effectively capture both the structural and temporal aspects of these dynamic graphs, dynamic graph neural networks have been developed. However, existing methods are usually tailored to process either continuous-time or discrete-time dynamic graphs, and cannot be generalized from one to the other. In this paper, we propose a decoupled graph neural network for large dynamic graphs, including a unified dynamic propagation that supports efficient computation for both continuous and discrete dynamic graphs. Since graph structure-related computations are only performed during the propagation process, the prediction process for the downstream task can be trained separately without expensive graph computations, and therefore any sequence model can be plugged-in and used. As a result, our algorithm achieves exceptional scalability and expressiveness. We evaluate our algorithm on seven real-world datasets of both continuous-time and discrete-time dynamic graphs. The experimental results demonstrate that our algorithm achieves state-of-the-art performance in both kinds of dynamic graphs. Most notably, the scalability of our algorithm is well illustrated by its successful application to large graphs with up to over a billion temporal edges and over a hundred million nodes.

## 1 INTRODUCTION

There are several complex networks in the real world, including social networks, transportation networks, biological networks, etc. In these networks, interactions between nodes include a great deal of valuable information, and graphs are regarded as good information

*Zhewei Wei is the corresponding author. The work was partially done at Gaoling School of Artificial Intelligence, Peng Cheng Laboratory, Beijing Key Laboratory of Big Data Management and Analysis Methods and MOE Key Lab of Data Engineering and Knowledge Engineering.

carriers for these complicated networks. As a result, a number of graph analysis problems arise, such as link prediction and anomaly identification. Due to their exceptional performance, Graph Neural Networks (GNNs) are recognized as effective tools for resolving these problems. However, most GNNs are designed for static graphs, while networks are constantly evolving over time. Focusing only on static graph information can result in missing crucial details, such as the patterns of network evolution. For example, social networks are characterized by continuous membership changes and shifts in following and unfollowing among users. By analyzing the temporal patterns of dynamic graphs, we can provide recommendations on potential friendships.

In recent years, various works have been developed to address the challenges of modeling and analyzing dynamic graphs. However, these algorithms are often designed for specific types of data. For example, it is challenging to adapt TGAT [40], which was established for Continuous-Time Dynamic Graphs (CTDGs), to Discrete-Time Dynamic Graphs (DTDGs) [47]. Similarly, methods developed for DTDGs, such as DySAT [28] and STGCN [41], cannot be directly applied to CTDGs. Although we could transform a CTDG into a sequence of snapshots taken at extremely short intervals and then use DTDG methods, the computational expense would be prohibitive.

**Motivation.** GNNs are important algorithms for solving graph-structured problems. The typical GNN layer consists of two modules, feature propagation and prediction, where feature propagation is the primary element impacting performance [39]. This has led to the development of decoupled GNNs. Several works, such as APPNP [16] and GBP [4], separate the feature propagation and nonlinear transformation operations, achieving significant improvements and scalability. By pre-calculating feature propagation, the need for complex computation during model training can be eliminated, saving time and effort. In addition, effective feature propagation can further improve the performance of GNNs. However, most of the existing decoupled models are tailored for learning static graphs, and their adaptation to dynamic graphs is challenging.

**Contribution.** Inspired by the decoupled static GNN, we propose a decoupled dynamic GNN method. The propagation progress takes dynamic graphs as input and generates temporal representations of all nodes in the graph. The model then trains for downstream tasks using the results of dynamic propagation, which no longer requires complicated graph computing at this stage and enables the utilization of arbitrary neural network models. Therefore, the computation for graph-structured data in dynamic graphs exists only in the propagation process, allowing for the construction of generic propagation methods for both continuous-time and discrete-time dynamic graphs.

We observe that CTDG methods, such as TGN [27], keep track of nodes affected by each graph event and adjust their embeddings, avoiding relearning the embeddings of all nodes and conserving
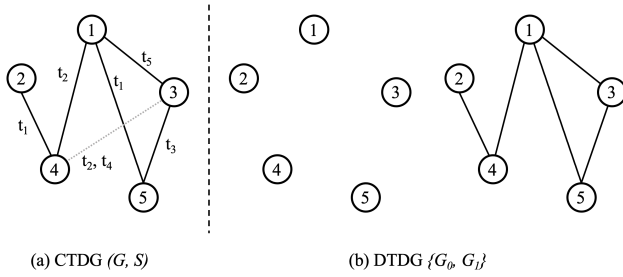
(a) CTDG *(G, S)*

(b) DTDG *{G₀, G₁}*

**Figure 1: Two types of dynamic graphs.**

computing resources. Since each snapshot is treated as a static graph in DTDG methods, edge deletion and the simultaneous occurrence of multiple graph events are naturally handled. Our objective is to develop a novel dynamic graph neural network that combines the strengths of both CTDG and DTDG methods. To achieve this, we introduce incremental node embedding update strategies specifically designed for handling batch graph events. This allows our model to process batch events similar to DTDG methods, while also keeping track of embedding changes akin to CTDG methods. Notably, our update strategy is not limited to adding new edges but also works seamlessly for removing edges. The main contributions can be summarized as follows:

- We propose a decoupled graph neural network for large dynamic graphs, which decouples the temporal propagation and prediction processes on dynamic graphs, enabling us to achieve great scalability and generate effective representation.
- We support the processing of continuous-time and discrete-time dynamic graphs by designing the generalized dynamic feature propagation. On the other hand, the model can fit various high-pass or low-pass graph filters to obtain a comprehensive temporal representation, by configuring various propagation formulas.
- Extensive experiments on seven benchmark datasets demonstrate the effectiveness of our method. Experimental results show that our model outperforms existing state-of-the-art methods. In addition, we evaluate our method on two large-scale graphs to show its excellent scalability.

## 2 NOTATIONS AND PRELIMINARY

In this section, we first introduce the necessary notations. Then we provide a concise overview of the classification of dynamic graphs and the common learning tasks associated with them.

**Notations.** A static graph is denoted as $G = (V, E)$, where $V$ is the set of $n$ nodes, and $E$ represents the set of $m$ edges. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ represent the adjacency matrix of $G$, with entry $\mathbf{A}(i, j) = w_{(i,j)} > 0$ being the weight of the edge between node $i$ and $j$, and $\mathbf{A}(i, j) = w_{(i,j)} = 0$ indicates non-adjacency. The degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix defined by $\mathbf{D}(i, i) = d(i) = \sum_{j \in V} w_{(i,j)}$. Each node $i \in V$ has a $d$-dimensional features vector $\mathbf{x}_i$, and all feature vectors form the feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$.

**Dynamic Graphs.** Dynamic Graphs can be summarized into two categories, CTDGs and DTDGs, depending on whether the entire timestamp is saved [12]. A CTDG is composed of an initial graph and a sequence of events, denoted as $(G, S)$, where $G$ is the initial state of the dynamic graph at time $t_0$ and $S$ is a set of observed

events on the graph. Each event consists of a triplet of *(event type, event, timestamp)*, where the *event type* can be edge additions, edge deletions, node additions, node deletions, node feature modifications, and so on. Therefore, $G_t$ is the new graph generated from the initial graph $G$ by sequentially completing the graph events of $\{t_1 \sim t\}$. Figure 1(a) shows an example of updating from an empty graph with only five nodes to the graph $G_5$ at time $t_5$, where the graph events involved are:

$$S = \{(AddEdge, (v_1, v_5), t_1), (AddEdge, (v_2, v_4), t_1),$$
$$(AddEdge, (v_1, v_4), t_2), (AddEdge, (v_3, v_4), t_2),$$
$$(AddEdge, (v_3, v_5), t_3), (DeleteEdge, (v_3, v_4), t_4),$$
$$(AddEdge, (v_1, v_3), t_5)\}.$$

A DTDG is represented as a sequence of snapshots, $\{G_0, \ldots, G_T\}$, which are sampled at regular time intervals. Figure 1(b) illustrates that the second snapshot, $G_1$, of the DTDG can be considered as the graph snapshot captured by the CTDG in Figure 1(a) at time $t_5$. However, it is important to note that the events occurring between $t_1$ and $t_5$ and their respective order are disregarded. Consequently, the DTDG fails to recognize the existence of the previous edge $(v_3, v_4)$ in the graph.

**Graph Learning Tasks.** Node classification and link prediction are traditional learning tasks for static graphs. We assume that each node is tagged with a label $\mathbf{Y}(i)$ from the label matrix $\mathbf{Y}$, but only the labels on a subset $V' \subset V$ are known. The objective of the node classification problem is to infer the unknown labels on $V \setminus V'$. In community detection, for instance, the label assigned to each node represents the community to which it belongs. Link prediction is the classical task of graph learning. It predicts whether an edge exists between two nodes that were not initially connected, inferring missing edges in $E$. In social networks, link prediction is also known as the friend recommendation task, predicting whether a user is interested in another.

Similarly, there are node-level and edge-level prediction tasks for dynamic graphs. Based on the historical information observed so far, we are able to accomplish dynamic node classification and future link prediction defined as follows.

DEFINITION 1 (DYNAMIC NODE CLASSIFICATION). *For a given graph $G_t = (V_t, E_t)$ and the incomplete label matrix $\mathbf{Y}'_t$, where $G_t$ can be regarded as the graph at timestamp $t$ in a CTDG or the $t$-th snapshot in a DTDG, and $\mathbf{Y}'_t$ associates a subset $V'_t \subset V_t$ with known class labels at timestamp/snapshot $t$, dynamic node classification is to classify the remaining nodes with unknown labels and estimate the label matrix $\mathbf{Y}_t$.*

DEFINITION 2 (FUTURE LINK PREDICTION). *For a given timestamp/snapshot $t$ and two nodes $i, j \in V_t$, future link prediction aims to predict whether edge $(i, j)$ will be generated in the next timestamp/snapshot or not, based on observations learned from all nodes and their links before timestamp $t$, i.e. observations of $\{G_0, \ldots, G_t\}$.*

## 3 RELATED WORKS

The Encoder-Decoder framework is a commonly used model in machine learning, which has been applied to various tasks such as unsupervised auto-encoder [9] and neural network machine translation models [31]. Recently, researchers have demonstrated that the Encoder-Decoder framework can generalize most high-performing dynamic graph learning algorithms [12, 47].

## 3.1 CTDGs Learning Methods

It is important to capture the changes in node embedding caused by every graph event when learning CTDGs. Most methods follow the training strategy that the encoder receives a sequence of graph events as input and reflects their influence in node embeddings. The decoder can therefore be a sequence learning model or a static network such as Multilayer Perceptron (MLP) or Support Vector Machine (SVM).

CTDNE [21] uses the temporal random walk as an encoder and designs three strategies for selecting the next-hop node in dynamic graphs. The introduction of temporal information reduces the uncertainty of embedding, resulting in better performance. The temporal point process is utilized by DyREP [32] to capture temporal changes at the node and graph levels. DyREP [32] builds embeddings of target nodes by aggregating information from neighboring nodes, where neighbors are limited by biasing the hop count selection of the temporal point process. These methods inherit the deficiencies of conventional graph representation learning methods, such as their inability to include node properties.

More prevalent CTDG learning encoders are based on Recurrent Neural Networks (RNNs), where the RNN generates memories from observed events associated with the target node via a memory function. The representative model TGN [27] comprises a memory component and an embedding component, where the memory component stores the historical memory of the given node. JODIE [19], DyREP [32], and TGAT [40] can be viewed as variants of TGN [27], and they differ in how they update embeddings and memories. Utilizing an asynchronous mail propagator, APAN [37] enforces that graph events are submitted to the model in timestamped order. Wang et al. [38] builds node representations using Causal Anonymous Walks (CAWs), which anonymize the node information on sampled temporal causal routes and apply attention learning to the sampled motifs. The resulting motifs are fed to RNNs to encode each walk as a representation vector. Subsequently, the representations of multiple walks are aggregated into a single vector using a self-attention process for downstream tasks.

Generally, methods specific to CTDGs efficiently learn node embeddings by tracking the impacted nodes for each graph event and updating their embeddings accordingly. However, these methods typically focus on considering the immediate neighbors linked with a graph event, such as the endpoints of an inserted edge, and few consider the impact on second-order neighbors [6]. Furthermore, the effect on higher-order neighbors or the overall graph is rarely evaluated, and there is limited discussion regarding edge deletions and simultaneous arrivals of multiple events.

## 3.2 DTDGs Learning Methods

In the DTDGs learning process, temporal patterns are measured by the sequential relationships between snapshots. Some works apply Kalman filtering [13, 29] or stacked spatial-temporal graph convolution networks (STGCN) [41] to create dynamic graph embeddings, and then use simple MLPs as decoders to perform the prediction task. More commonly, static methods, such as GAE and VGAE [14], are used to generate node embeddings of each snapshot. The embeddings are then sorted by time and treated as sequential data, and a sequential decoder is applied to extract the temporal patterns from them.

To obtain embeddings of each snapshot, GraRep [3], HOPE [22], and M-NMF [36] construct encoders using matrix decomposition, while DeepWalk [26] and node2vec [8] transform the graph structure into node-level embeddings using random walk. These algorithms, however, are shallow embedding methods, meaning that they do not consider the attribute information of the graph. Also, there is no parameter sharing between nodes, which makes these methods computationally inefficient. Graph neural networks are efficient ways for learning both the structure and attribute information of a graph. GNNs follow the message-passing framework, in which each node generates embeddings by aggregating information of neighbors [7]. To improve the efficiency, Graph Convolutional Network (GCN) [15] derives the layer-by-layer propagation formula from the first-order approximation of the localized spectral filters on the graph:

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right) \tag{1}$$

where $\mathbf{A}$ and $\mathbf{D}$ are the adjacency matrix and degree matrix, respectively. $\mathbf{W}^{(\ell)}$ is the learnable parameter of layer $\ell$, and $\sigma$ is a nonlinear activation function such as ReLU. $\mathbf{H}^{(\ell)}$ is the learnt node representation at the $\ell$-th layer, and $\mathbf{H}^{(0)} = \mathbf{X}$. AddGraph [44] employs GCN as the encoder to analyze the structural information of each snapshot, while a sequence decoder is used to determine the relationships between snapshots. Graph Attention Network (GAT) [34] is an attention mechanism based on GCN that assigns various weights to the features of neighbors via weighted summation. DyGAT [28] employs GAT as an encoder for DTDGs learning, and node embeddings are generated by jointly computing self-attentions of neighborhood structure and time dimensions.

Long Short-Term Memory (LSTM) [10] is a widely used sequence model, known for its ability to effectively capture long-term temporal dependencies and correlations. Therefore, Seo et al. [30] and Manessi et al. [20] use LSTM as decoders, and their encoders are GCNs or their different versions. The combination structure of GNNs and LSTMs has demonstrated its efficacy in object detection [43] and pandemic forecasting [23] areas. EvolveGCN [25] uses LSTM and Gate Recurrent Unit (GRU) to update the GCN's parameters at each snapshot since it focuses on the evolution of the GCN's parameters rather than the node representation at each snapshot. These DTDG-specific methods typically treat each snapshot as a static graph, making it easy for them to address edge deletion and many simultaneous edges. However, they are unable to track the particular impact of each graph event on node embedding, and the recomputation of each snapshot is computationally expensive.

## 4 DECOUPLED GRAPH NEURAL NETWORK

**Overview.** As previously indicated, we aim to design a decoupled GNN with high scalability for dynamic graphs. In addition, we also require the model to operate multi-event arrivals simultaneously and support edge deletion while keeping tracking changes in node embedding, which incorporates the benefits of the CTDGs-specific and DTDGs-specific models. Therefore, inspired by the scalable static GNN framework [2, 39], we develop a decoupled GNN for large dynamic graphs, in which the dynamic propagation of the graph is decoupled from the prediction process. To enable efficient computation on large-scale dynamic graphs, we employ

**Algorithm 1:** GENERALPROPAGATION

---

**Input:** Graph $G$, weight coefficients $\gamma_k$, convolutional
coefficients $\beta$, threshold $r_{max}$, initialized $(\hat{\pi}, r)$

1 **while** *exist* $i \in V$ *with* $|r(i)| > r_{max} \cdot d(i)^{1-\beta}$ **do**
2     $\hat{\pi}(i) \leftarrow \hat{\pi}(i) + \gamma_0 \cdot r(i)$;
3     **for** *each* $j \in N(i)$ **do**
4        $r(j) \leftarrow r(j) + \frac{\gamma \cdot w_{(i,j)} \cdot r(i)}{d(i)^{1-\beta} d(j)^{\beta}}$ ;
5     $r(i) \leftarrow 0$;
6 **return** $(\hat{\pi}, r)$;

---

dynamic propagation with strict error guarantees (as described in Section 4.1). This approach eliminates learning parameters in the propagation process, facilitating independent graph propagation for generating temporal representations of all nodes. The prediction process focuses on learning the underlying graph dynamics from the representations of nodes, which does not contain expensive graph computations, enabling the use of arbitrary learning models, as described in Section 4.2.

**Scalable GNNs.** In order to improve the scalability of GNN models, a line of research tries to decouple the propagation and prediction of conventional GNN layers. The idea behind them is to apply MLPs to batches of nodes simply without taking the graph structure into account, which is proposed by SGC [39] first. For implementation, the representation matrix $\mathbf{Z}$ is generated first following this general formulation propagation:

$$\mathbf{Z} = \sum_{k=0}^{\infty} \gamma_k (\mathbf{D}^{-a} \mathbf{A} \mathbf{D}^{-b})^k \mathbf{X}, \tag{2}$$

where $\mathbf{X}$ denotes the input feature matrix, $a$ and $b$ are convolution coefficient, $\gamma_k (k = 0, 1, 2, \dots)$ is the weight of the $k$-th step convolution. When $a = b = \frac{1}{2}$ and $\gamma_k = 1$, Equation 2 can be considered a GCN with an infinite number of layers, i.e., a stack of infinite layers of Equation 1. However, the parameters of each layer are discarded for better scalability. Therefore, MLPs take the representation matrix $\mathbf{Z}$ as input and trains for downstream tasks. Mini-batch training can be easily accomplished since node representations can be viewed as distinct input samples for the neural network. Numerous models, including APPNP [16], SGC [39], and GBP [4], can be regarded as versions of Equation 2 constructed by choosing different values for $a$, $b$, and $\gamma_k$. By varying $\gamma_k$, Equation 2 can approximate any form of graph filter. For instance, Equation 2 corresponds to a low-pass graph filter when all $\gamma_k (k = 0, 1, 2, \dots)$ satisfy $\gamma_k \geq 0$, and Equation 2 relates to a high-pass filter when $\gamma_k$ is of the form $(-\alpha)^k$ with $\alpha \in (0, 1)$. For simplicity, we assume that $a = \beta$, $b = 1 - \beta$, and the sequence of $\gamma_k$ is a geometric progression with a common ratio $\gamma = \frac{\gamma_{k+1}}{\gamma_k}$ and $0 < |\gamma| < 1$ in this paper.

We aim to extend the previous concept to dynamic graphs. Firstly, we derive temporal representations for all nodes in the graph based on dynamic approximate propagation, which can be efficiently precomputed. Next, we batch the structurally enhanced temporal representations of nodes and feed them into the learning model. This decoupling framework, derived from scalable static GNNs, permits the use of any sequence model while preserving high scalability.

**Approximate propagation.** The summation in Equation 2 goes to infinity, which makes it computationally infeasible. Following PPRGo [2] and AGP [35], we consider its approximate version. By representing each dimension of the feature matrix as an $n$-dimensional vector $x$, the feature matrix can be turned into a sequence of $\{x_0, \dots, x_{d-1}\}$, where the propagation of each vector is conducted independently. Equation 2 can therefore be expressed in a equivalent vector form: $\pi = \sum_{k=0}^{\infty} \gamma_k (\mathbf{D}^{-\beta} \mathbf{A} \mathbf{D}^{\beta-1})^k x$. As illustrated in Algorithm 1, we generalize the propagation algorithm [45] to a weighted version to support weighted graph neural networks and relax the requirement for positive weight coefficients. We denote the approximate solution as $\hat{\pi}$, and the cumulative error of all steps is denoted as $r$. For initialization, we set $\hat{\pi} = 0$ and $r = x$. The propagation starts from the node whose residual exceeds the error tolerance $r_{max}$. Then, the node distributes an equal portion of its residual to its neighboring nodes, and the remainder is transformed into its estimate to record the amount of information already propagated by that node. The feature propagation concludes when the residuals of all graph nodes satisfy the error bound.

The neural network model receives the structurally improved feature matrix $\hat{\mathbf{Z}} = (\hat{\pi}_0, \dots, \hat{\pi}_{d-1})$ as input and is trained to get the final representation of the nodes based on the subsequent task. For instance, the multi-label node classification task typically uses $\mathbf{Y} = softmax(MLP(\hat{\mathbf{Z}}))$. By decoupling propagation and prediction, the model training complexity is independent of the graph topology, which enhances training efficiency and enables the use of sophisticated prediction networks simultaneously.

### 4.1 Dynamic Propagation

We consider a dynamic graph $\mathcal{G} = \{G_0, G_1, \dots, G_T\}$, where each $G_t (t \in [0, T])$ is the graph derived from the initial graph in a CTDG after finishing the graph events before timestamp $t$, or the $t$-th snapshot in a DTDG. That is, $G_t$ refers to the $t$-th observed status of the dynamic graph. We are not concerned with how $G_t$ is obtained, i.e. how the dynamic graph $\mathcal{G}$ is stored. The overall update procedure is summarized in Algorithm 2. We obtain the feature propagation matrix for each $G_t$, and $\mathbf{Z}_t$ is derived iteratively from $\mathbf{Z}_{t-1}$, as in lines 9-19. The estimated vector $\hat{\pi}$ and residual vector $r$ inherit the propagation results from the previous time step and make the necessary updates based on the current graph structure.

To construct sequential representations for all nodes in the dynamic graph, it is necessary to comprehend how to quantify the impact that changes to the network have on every node. Therefore, each node should have its individual observation perspective, with a unique comprehension of each graph modification. To improve the computational efficiency, we propose to incrementally compute the node representation when the graph changes. We start with the following theorem on invariant properties. Due to the page limit, we defer the proof to the technical report [1].

THEOREM 1 (THE INVARIANT PROPERTY). *Suppose $\hat{\pi}(i)$ is the estimate of node $i$, $r(i)$ is its residual, and $x(i)$ is its input feature, for each node $i \in V$, we notice that $\hat{\pi}(i)$ and $r(i)$ satisfy the invariant property as follow:*

$$\hat{\pi}(i) + \gamma_0 r(i) = \gamma_0 x(i) + \sum_{j \in N(i)} \frac{\gamma \cdot w_{(i,j)} \cdot \hat{\pi}(j)}{d(i)^{\beta} d(j)^{1-\beta}}. \tag{3}$$

**Algorithm 2:** DynamicPropagation

---

**Input:** Dynamic graph $\mathcal{G}$, weight coefficients $\gamma_k$, convolutional coefficients $\beta$, threshold $r_{max}$, feature matrix $\mathbf{X}^{n \times d}$

1 **parallel for** *each column* $x \in \mathbf{X}$ **do**
    /* Step 1. Generate estimated and residual vector $\hat{\pi}$, $r$ for the initial graph $G$ */
2     $\hat{\pi} \leftarrow 0, r \leftarrow x$ ;
3     $\hat{\pi}, r \leftarrow$ GeneralPropagation$(G, \gamma_k, \beta, r_{max}, \hat{\pi}, r)$ ;
4     **for** *each time step* $t \in [1, T]$ **do**
5         Updating $G$ with graph events at time $t$;
6         Collect the affected nodes at time $t$ as $V_A$ ;
        /* Step 2. Maintain the estimated and residual vector $\hat{\pi}$, $r$ in accordance with the invariant property. */
7         **parallel for** *each* $u \in V_A$ **do**
8             $\hat{\pi}(u) \leftarrow \hat{\pi}(u) \cdot \frac{d(u)_t^{1-\beta}}{d(u)_{t-1}^{1-\beta}}$ ;
9             $r(u) \leftarrow r(u) + \hat{\pi}(u) \cdot \frac{d(u)_{t-1}^{1-\beta} - d(u)_t^{1-\beta}}{\gamma_0 \cdot d(u)_t^{1-\beta}}$ ;
10         **parallel for** *each* $u \in V_A$ **do**
11             $\Delta r(u) \leftarrow (\hat{\pi}(u) + \gamma_0 r(u) - \gamma_0 x(u)) \cdot \frac{d(u)_{t-1}^{\beta} - d(u)_t^{\beta}}{d(u)_t^{\beta}}$ ;
12             **for** *each* $v \in N_{add,t}(u)$ **do**
13                 $\Delta r(u) \leftarrow \Delta r(u) + \frac{\gamma \hat{\pi}(v)}{d(u)_t^{\beta} d(v)_t^{1-\beta}}$ ;
14             **for** *each* $v \in N_{delete,t}(u)$ **do**
15                 $\Delta r(u) \leftarrow \Delta r(u) - \frac{\gamma \hat{\pi}(v)}{d(u)_t^{\beta} d(v)_t^{1-\beta}}$ ;
16             $\Delta r(u) \leftarrow \Delta r(u) / \gamma_0$ ;
17             $r(u) \leftarrow r(u) + \Delta r(u)$ ;
        /* Step 3. Propagation on the graph at time $t$. */
18         $\hat{\pi}, r \leftarrow$ GeneralPropagation$(G, \gamma_k, \beta, r_{max}, \hat{\pi}, r)$ ;

19 **return** Embedding matrix $\hat{\mathbf{Z}}^{n \times d} = (\hat{\pi}_0, \dots, \hat{\pi}_{d-1})$ ;

---

**Generalized update rules.** Without loss of generality, we assume that an edge $(u, v)$ with weight $w_{(u,v)}$ is inserted to the graph. According to Equation 3, the set of affected nodes is $V_A = \{u, w | w \in N(u)\}$. For node $u$, the increment caused by the insertion can be quantified as $(\hat{\pi}(u) + \gamma_0 r(u) - \gamma_0 x(u)) \frac{d(u)^{\beta} - (d(u) + w_{(u,v)})^{\beta}}{\gamma_0 \cdot (d(u) + w_{(u,v)})^{\beta}} + \frac{\gamma w_{(u,v)} \hat{\pi}(v)}{\gamma_0 (d(u) + w_{(u,v)})^{\beta} d(v)^{1-\beta}}$, since the degree is updated to $d(u) + w_{(u,v)}$ and a new neighbor $v$ appears. According to the meaning of estimate and residual, we add this increment to the residual of node $u$. Similarly, for each node $w \in N(u)$, $\frac{\hat{\pi}(u)}{d(u)^{1-\beta}}$ in its equation will be updated to $\frac{\hat{\pi}(u)}{(d(u) + w_{(u,v)})^{1-\beta}}$ as a result of the change of node $u$'s degree. To guarantee that the update time complexity of each insertion is $O(1)$, the following updates are performed to prevent alterations to node $u$'s neighbors:

- $\hat{\pi}(u) = \frac{d(u)^{1-\beta}}{(d(u) + w_{(u,v)})^{1-\beta}} \cdot \hat{\pi}(u)$;

- $r(u) = r(u) + \frac{\hat{\pi}(u)}{\gamma_0} \cdot \left( \frac{d(u)^{1-\beta}}{(d(u) + w_{(u,v)})^{1-\beta}} - 1 \right)$.

The detailed calculation process of the update and its batched version can also be found in the technical report [1]. Since none of variables involved in the equation of other nodes have changed, the increment induced by the insertion of the edge $(u, v)$ is zero from the perspective of node $i \in V, i \neq u$ and $i \notin N(u)$. Algorithm 1 is then used to propagate this increment from node $u$ to its neighbors, informing other nodes of the change in the graph.

The preceding procedure can be easily generalized to the case of deleting the edge $(u, v)$ with weight $w_{(u,v)}$ by simply replacing $(d(u) + w_{(u,v)})$ with $(d(u) - w_{(u,v)})$. Therefore, it is unnecessary to recalculate the feature propagation when the graph changes, but rather obtain the current propagation matrix incrementally based on the past calculation result. In addition, we have Theorem 2 to guarantee the error of propagation on each $G_t$.

THEOREM 2 (ERROR ANALYSIS). *Suppose $\hat{\pi}_t(i)$ is the estimate of node $i$ at time $t$, $\pi_t(i)$ is its ground-truth estimate at time $t$, $d(i)_t$ is its degree at time $t$, and $r_{max}$ is the error threshold, for each node $i \in V$, we have $|\pi_t(i) - \hat{\pi}_t(i)| \leq r_{max} \cdot d(i)_t^{1-\beta}$ holds for $\forall t \in \{0, 1, \dots, T\}$.*

**Handle CTDGs.** We can utilize the aforementioned update strategy to handle incoming graph events accompanied by either inserting or removing edges. For each arriving edge $(u, v)$, we can ensure that Equation 3 holds at all nodes by updating only the estimate and residual at node $u$, and the time complexity of the update is $O(1)$. Therefore, the above update strategy can be well adapted to sequences of frequently arriving graph events in CTDGs.

**Handle DTDGs.** For two successive snapshots $G_{t-1}$ and $G_t$ in a DTDG, we regard the changes between the two snapshots as graph events arrived simultaneously, and it is easy to statistically extract $V_A$. We can then compute exactly the increment between the two snapshots by substituting $w_{(u,v)}$ with the degree change $\Delta d(u) = d(u)_t - d(u)_{t-1}$ for each affected node $u \in V_A$. Similarly, Algorithm 1 transmits information about the changes in the graph to other nodes. Therefore, when a new snapshot $G_t$ arrives, we incrementally update the feature propagation matrix based on the feature propagation results of snapshot $G_{t-1}$. Since batches of graph updates can be efficiently processed, our method naturally supports DTDGs and maintains tracking for underlying node embeddings.

**Remark.** In comparison to CTDG-specific methods, our approach updates the dynamic graph based on timestamps rather than relying solely on the order of each edge. Since it is common for multiple graph events to occur simultaneously at a single time step in real-world scenarios, handling each event individually would be suboptimal. In contrast to DTDG-specific methods, where each graph snapshot is treated as a static graph, we adopt an incremental update approach, which allows us to update the graph incrementally based on the differences between two successive snapshots. Based on this strategy, we avoid recalculating the underlying node embeddings for each snapshot and disregarding changes to them.

## 4.2 Prediction

In this section, we provide illustrations of the prediction phase by considering dynamic node classification and future link prediction as examples. These two tasks are defined in detail in Section 2.

**Dynamic node classification.** We can incrementally obtain the feature propagation matrix at each time $t$ using Algorithm 2. Each

row of the feature propagation matrix $\hat{Z}_t$, denoted as a $d$-dimensional vector $z_{t,i}$, is the structural-enhanced node representation vector of node $i \in V$ at time $t$. Therefore, $z_{t,i}$ is the representation vector of node $i$ under the error tolerance control described in Theorem 2. Since the aggregation of node features based on graph structure has been completed during the propagation process, the node representation vector $z_{t,i}(t=1,\ldots,T)$ for each node $i \in V$ can be regarded as the standard input vector of neural networks at this stage. For instance, we use a two-layer MLP to predict the label of node $i$ at time $t$ as: $Y_t(i) = softmax(MLP(z_{t,i}))$.

**Future link prediction.** In this task, we aim to learn the temporal pattern of each node to forecast if the two given nodes would be linked at the given time. The changes in the node representation over time can be regarded as a time series, and the temporal information contained within it can be captured by a common temporal model such as LSTM. Notice that utilizing sequence $\{z_{1,i}, \ldots, z_{T,i}\}$ to describe the dynamic network will provide a very subjective impression from node $i$. As a result, when changes in the graph have a large influence on node $i$, the representation vector $z_{t,i}$ changes significantly with respect to the previous moment $z_{t-1,i}$. The vector changes little from that of the previous state $z_{t-1,i}$ when changes in the graph have little influence on node $i$. Note that the degree of influence is related to the final feature propagation matrix generated by Algorithm 2. Therefore, node $i$'s perception of the degree of graph change is influenced by the descriptions of its neighboring nodes through the propagation process. The completion of the future link prediction task involves the following three steps.

- **Firstly**, we calculate the difference between each node's state in two consecutive graph states as $\delta_{t,i} = g(z_{t,i}, z_{t-1,i})$, where $g(\cdot)$ is distance measure function. We implement $g(\cdot)$ as a simple first-order distance, although it could also be a $\ell_2$-norm, cosine similarity, or other complicated design. Based on the above, we interpret $\delta_{t,i}(s)$ as the score of graph changes from the perspective of node $i$ at the $s$-th feature dimension.
- **Secondly**, sequence models, such as LSTM, directly take the sequence $\{\delta_{1,i}, \ldots, \delta_{t,i}\}$ as input to capture the temporal patterns for node $i$. Since the graph structure information is already included in $z_{t,i}$ and $\delta_{t,i}$, the sequence model can be employed more effectively by focusing solely on temporal patterns. The predicted state at time $t$ is denoted as $h_t = \mathcal{M}(h_{t-1}, \delta_t)$, where $\mathcal{M}$ is the chosen sequence learning model, $\delta_t$ is the current input vector, and $h_{t-1}$ is the learned prior state. The standard LSTM cell is defined by the following formula:

$$
\begin{aligned}
i_t &= \sigma(W_i h_{t-1} + U_i \delta_t + b_i), \\
f_t &= \sigma(W_f h_{t-1} + U_f \delta_t + b_f), \\
o_t &= \sigma(W_o h_{t-1} + U_o \delta_t + b_o), \\
\tilde{c}_t &= \tanh(W_c h_{t-1} + U_c \delta_t + b_c), \\
c_t &= f_t \odot C_{t-1} + i_t \odot \tilde{c}_t, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
\tag{4}
$$

where $\sigma$ is the sigmoid activation function, $\odot$ denotes the matrix product operation, $i_t$, $f_t$ and $o_t$ represent the degree parameters of the input gate, forgetting gate and output gate of the LSTM cell at time $t$. $\{W_i, U_i, b_i\}$, $\{W_f, U_f, b_f\}$, $\{W_o, U_o, b_o\}$ are their corresponding network parameters, respectively. $\tilde{c}_t$ denotes the candidate states used to update the cell states. $\{W_c, U_c, b_c\}$ are the parameters of the network for generating candidate memories. $c_t$ is formed as the output vector $h_t$ at the current time $t$ after the output gate has discarded some information. Note that the LSTM cell could be replaced by a GRU cell or a Transformer cell, as $\mathcal{M}$ is free from graph-related computations.

- **Finally**, we combine the pair of hidden states of node $i$ and $j$ as $\varphi_t(i, j) = f(h_{t,i}, h_{t,j})$, where $f(\cdot)$ is the combine function, and we experiment on concatenation following previous work [27, 42]. Then the probability score of edge $(i, j)$'s existence at time $t$ is given by $Y_t(i, j) = \sigma(MLP(\varphi_t(i, j)))$.

## 5 EXPERIMENTS

In this section, we evaluate the effectiveness of our method against state-of-the-art methods on two representative tasks, future link prediction and dynamic node classification, on both CTDGs and DTDGs. Furthermore, we conduct experiments on two large-scale dynamic graphs to demonstrate the scalability of our method.

**Datasets.** We conducted experiments on seven real-world datasets, including Wikipedia [19], Reddit [19], UCI-MSG [24], Bitcoin-OTC [17, 18], Bitcoin-Alpha [17, 18], GDELT [46] and MAG [11, 46]. The statistics of datasets are presented in Table 1. In all graphs, the weight of an edge is determined by its frequency of occurrence. More details about the datasets can be found in the technical report [1].

**Baseline methods.** We compare our method to state-of-the-art dynamic graph neural networks, including TGN [27], CAW-Ns [38] for CTDGs and ROLAND [42] for DTDGs. In the two CTDG datasets, Wikipedia and Reddit, we strictly inherit the baseline results from their papers and follow the experimental setting of TGN [27]. In the three DTDG datasets, UCI-Message, Bitcoin-Alpha, and Bitcoin-OTC, our experimental setting is closely related to those of EvolveGCN [25] and ROLAND [42], and we adopt the original paper's stated results. To provide a fair comparison, we employ the same data processing and partitioning techniques as TGN [27] and ROLAND [42]. For the two large-scale datasets GDELT and MAG, we utilize the results reported by TGL [46]. Other baseline methods are described in Section 3.

### 5.1 Experiments on CTDGs

**Experimental Setting.** We conduct experiments on Wikipedia and Reddit dataset in both transductive and inductive settings, following [27]. In both settings, the first 70% of edges are used as the training set, 15% are used as the validation set, and the remaining 15% are used as the test set. In the transductive setting, we predict future links for observed nodes in the training set. In the inductive setting, the future linking status of nodes that do not present in the training set is predicted. We formulate the prediction of future links between two nodes as a binary classification problem. More specifically, we assign a label of 1 to indicate that the two nodes will be connected in the future, while a label of 0 signifies that there will be no link between them. The time span of the prediction is one time step. The popular classification metric Average Precision (AP) is employed to evaluate the algorithm's performance on both future link prediction and dynamic node classification tasks. In order to maintain the balance of the data, we generate one negative sample for each test edge or node when computing AP, following the experimental setting in TGN [27] and TGL [46]. For our method,

**Table 1: Statistics of the datasets.**

|  | #nodes | #edges | max($t$) | #classes | #node features | #edge features |
|---|---|---|---|---|---|---|
| Wikipedia | 9,227 | 157,474 | 152,757 | 2 | 172 (random) | 172 |
| Reddit | 11,000 | 672,447 | 669,065 | 2 | 172 (random) | 172 |
| UCI-Message | 1,899 | 59,835 | 87 | - | 128 (random) | - |
| Bitcoin-OTC | 5,881 | 35,592 | 138 | - | 128 (random) | 1 |
| Bitcoin-Alpha | 3,783 | 24,186 | 138 | - | 128 (random) | 1 |
| GDELT | 16,682 | 191,290,882 | 170,522 | 81 | 413 | 186 |
| MAG | 121,751,665 | 1,297,748,926 | 120 | 152 | 768 | - |

**Table 2: Future link prediction on CTDGs. AP (%) ± standard deviations computed of 10 random seeds are exhibited.**

|  | Wikipedia | | Reddit | |
|---|---|---|---|---|
|  | Transductive | Inductive | Transductive | Inductive |
| GAE | 91.44 ± 0.1 | - | 93.23 ± 0.3 | - |
| VGAE | 91.34 ± 0.3 | - | 92.92 ± 0.2 | - |
| DeepWalk | 90.71 ± 0.6 | - | 83.10 ± 0.5 | - |
| Node2Vec | 91.48 ± 0.3 | - | 84.58 ± 0.5 | - |
| GAT | 94.73 ± 0.2 | 91.27 ± 0.4 | 97.33 ± 0.2 | 95.37 ± 1.1 |
| GraphSAGE | 93.56 ± 0.2 | 91.09 ± 0.3 | 97.65 ± 0.2 | 96.27 ± 0.2 |
| CTDNE | 92.17 ± 0.5 | - | 91.41 ± 0.3 | - |
| Jodie | 94.62 ± 0.5 | 93.11 ± 0.4 | 97.11 ± 0.3 | 94.36 ± 1.1 |
| TGAT | 95.34 ± 0.1 | 93.99 ± 0.3 | 98.12 ± 0.2 | 96.62 ± 0.3 |
| DyRep | 94.59 ± 0.2 | 92.05 ± 0.3 | 97.98 ± 0.1 | 95.68 ± 0.2 |
| TGN | 98.46 ± 0.1 | 97.81 ± 0.1 | 98.70 ± 0.1 | 97.55 ± 0.1 |
| CAW-N-mean | 98.82 ± 0.1 | 98.28 ± 0.1 | 98.72 ± 0.1 | 98.74 ± 0.1 |
| CAW-N-attn | 98.84 ± 0.1 | 98.31 ± 0.1 | 98.80 ± 0.1 | 98.77 ± 0.1 |
| **ours** | **99.16 ± 0.3** | **98.54 ± 0.2** | **99.51 ± 0.5** | **98.81 ± 0.6** |

**Table 3: Dynamic node classification on CTDGs. ROC AUCs (%) are exhibited.**

|  | Wikipedia | Reddit |
|---|---|---|
| Jodie | 81.37 | **70.91** |
| DySAT | 86.30 | 61.70 |
| TGAT | 85.18 | 60.61 |
| TGN | 88.33 | 63.78 |
| APAN | 82.54 | 62.00 |
| **ours** | **89.81** | 67.53 |

we set the weight coefficients $\gamma_k = \alpha(1-\alpha)^k$, which is known as the Personalized PageRank weights with a hyperparameter $\alpha \in (0, 1)$. The standard LSTM is utilized as the sequence model to learn the temporal patterns present in the node representation. Since no node features are provided, we use a randomly generated 172-dimensional vector as the initial node feature vector.

**Results.** The results of future link prediction in both transductive and inductive settings are shown in Table 2. The presented results are the average of 10 runs. Our method performs better than baseline methods in both transductive and inductive settings. The interesting thing is that we did not use the provided edge features and achieve comparable or even better performance. This may be strongly related to the experimental setting and dataset. For the current future link prediction, we simply need to forecast whether a connection will be created between two given nodes in the future.

The specifics of that link are practically of no concern. The publicly accessible edge features of Wikipedia and Reddit are derived from the textual content of each edit or post on the respective web page and sub-reddit. The learning objective is to detect whether a user would edit a certain page or post on a given sub-reddit in the future, without predicting the edit or post content. It is possible that semantic information of textual material is superfluous. The historical interaction data already contains sufficient information to reveal users' preferences for particular pages and sub-reddits. Our hypothesis is also supported by the results of our method on graphs that lack semantic information.

Table 3 shows the experimental results for the dynamic node classification. For node classification, we always use the most recent node representation based on the history observed so far. The three-layer MLP is employed as the classifier. The results in Table 3 show that our method effectively captures the temporal changes of the nodes in time, thus enabling the correct classification of the nodes.

## 5.2 Experiments on DTDGs

**Experimental Setting.** We use three datasets in this experiment: Bitcoin-OTC, Bitcoin-Alpha and UCI-Message. To ensure a fair comparison, we partition the dataset and calculate evaluation measures in the same manner as ROLAND [42]. Since node features and edge features are not provided in these three datasets, we generate the 128-dimensional random vector to serve as the initial node feature. The ranking metric, Mean Reciprocal Rank (MRR), is employed to evaluate performance. We collect 1000 negative samples for each positive sample and then record the ranking of positive samples according to predicted probabilities. MRR is calculated independently for each snapshot in the test set, and the average of all snapshots is reported. For our method, we combine the node temporal representations obtained under settings $\gamma_k = \alpha(1-\alpha)^k$ and $\gamma_k = \alpha(\alpha-1)^k$ to approximate the low-pass and high-pass filters on the graph and introduce low-frequency and high-frequency information, respectively. Three traditional sequence models, LSTM [10], GRU [5] and Transformer [33], are used to finish the future link prediction task.

**Results.** The results in Table 4 demonstrate the state-of-the-art performances of our method. In the Bitcoin-OTC dataset, our method outperforms the second-best method ROLAND by 41%. The results show that the LSTM model consistently outperforms the GRU model, which is possibly due to the LSTM having more parameters. The Transformer model tends to achieve a higher MRR, which could be attributed to its holistic approach to the temporal sequence of nodes and reduced reliance on previous hidden states. Additionally, we conducted an ablation study in the technical report [1] to validate the necessity of introducing high-frequency information.

**Table 4: Future link prediction on DTDGs. MRR ± standard deviations computed of 3 random seeds are exhibited.**

|  |  | UCI-Message | Bitcoin-Alpha | Bitcoin-OTC |
|---|---|---|---|---|
|  | GCN | 0.1141 | 0.0031 | 0.0025 |
|  | DynGEM | 0.1055 | 0.1287 | 0.0921 |
|  | dyngraph2vecAE | 0.0540 | 0.1478 | 0.0916 |
|  | dyngraph2vecAERNN | 0.0713 | 0.1945 | 0.1268 |
|  | EvolveGCN-H | 0.0899 | 0.1104 | 0.0690 |
|  | EvolveGCN-O | 0.1379 | 0.1185 | 0.0968 |
|  | ROLAND Moving Average | 0.0649 ± 0.0049 | 0.1399 ± 0.0107 | 0.0468 ± 0.0022 |
|  | ROLAND MLP | 0.0875 ± 0.0110 | 0.1561 ± 0.0114 | 0.0778 ± 0.0024 |
|  | ROLAND GRU | 0.2289 ± 0.0618 | 0.2885 ± 0.0123 | 0.2203 ± 0.0167 |
|  | GRU | 0.2024 ± 0.0010 | 0.3289 ± 0.0070 | 0.2985 ± 0.0121 |
| **ours** | LSTM | 0.2140 ± 0.0034 | **0.3405 ± 0.0133** | 0.3102 ± 0.0046 |
|  | Transformer | **0.2314 ± 0.0048** | 0.3173 ± 0.0135 | **0.3110 ± 0.0049** |

**Table 5: Dynamic node classification on large graphs. F1-Micros (%) are exhibited.**

|  | GDELT | MAG |
|---|---|---|
| Jodie | 11.25 | 43.94 |
| DySAT | 10.05 | 50.42 |
| TGAT | 10.04 | 51.72 |
| TGN | 11.89 | 49.20 |
| APAN | 10.03 | - |
| **ours** | **25.49** | **61.40** |

## 5.3 Experiments on Large Graphs

**Experimental Setting.** To demonstrate the scalability of our method, we conduct experiments on two large-scale real-world graphs, GDELT and MAG. We exclude EvolveGCN [25], ROLAND [42] and CAW-Ns [38] from experiments on GDELT and MAG datasets, since they met out-of-memory issues on both datasets. Note that the scalability of the baseline methods JODIE [19], DySAT [28], TGAT [40], TGN [27], and APAN [37] was not taken into account in their original papers, and their original versions cannot be trained on these two large-scale dynamic graphs. TGL [46] has successfully applied these methods to large-scale graphs by developing a distributed dynamic graph neural network training framework. In contrast, our method can learn large-scale graphs directly. Our method exhibits greater scalability due to the elimination of parameters in the propagation process, allowing the training on these two large-scale graphs to be completed on a single machine. We validate the performance of all methods on the dynamic node classification task and compare their performance using the multiple-class classification metric F1-Micro. To guarantee a fair comparison, we ensure that the training, validation and test sets are consistent with the settings in TGL. For our method, we set $\gamma_k = \alpha(1-\alpha)^k$ to obtain the temporal representation of each node, and a three-layer MLP is utilized to complete the training for the classification task.

**Results.** Table 5 shows the dynamic node classification results. Compared to baseline methods, we achieve significant performance improvement in both datasets. Specifically, our method improves F1-Micro by 13.6 on the GDELT dataset and 9.68 on the MAG dataset. This indicates that our method can effectively capture the dynamic changes in node representations by precisely locating the directly affected nodes via Equation 3 and quantifying the degree

of graph change. The following propagation process that immediately follows broadcasts the change from the affected node to its surroundings, so that higher-order neighbors can also naturally perceive the change on the graph. However, from a practical application standpoint, the performance of all methods in GDELT is not adequate. We note that this is due to the presence of much noise in the labeled data of GDELT. Participants can participate in events held around the world via remote means such as online, resulting in some nodes may simultaneously belong to many classes.

## 6 CONCLUSION

This paper proposes a universal general graph neural network for dynamic graphs that can extract the structural and attribute information of the graph, as well as the temporal information. Our algorithm is based on the framework of decoupled GNNs, which can pre-compute temporal propagation in dynamic graphs and then train them for downstream tasks depending on the nodes' temporal representation. We devised a unified dynamic propagation method to support the learning on both continuous-time and discrete-time dynamic graphs. Empirical studies on continuous-time and discrete-time dynamic graphs at various scales demonstrate the scalability and state-of-the-art performance of our algorithm.

# REFERENCES

[1] http://arxiv.org/abs/2305.08273.

[2] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *KDD*, pages 2464–2473, 2020.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900, 2015.

[4] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Scalable graph neural networks via bidirectional propagation. In *NeurIPS*, 2020.

[5] Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] Zeyu Cui, Zekun Li, Shu Wu, Xiaoyu Zhang, Qiang Liu, Liang Wang, and Mengmeng Ai. Dygcn: Efficient dynamic graph embedding with graph convolutional network. *TNNLS*, 2022.

[7] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272, 2017.

[8] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[9] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[12] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[13] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *STOC*, pages 504–513, 2000.

[14] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

[17] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *WSDM*, pages 333–341, 2018.

[18] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *ICDM*, pages 221–230, 2016.

[19] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Learning dynamic embeddings from temporal interactions. *arXiv preprint arXiv:1812.02289*, 2018.

[20] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[21] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW*, pages 969–976, 2018.

[22] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, pages 1105–1114, 2016.

[23] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer graph neural networks for pandemic forecasting. In *AAAI*, volume 35, pages 4838–4845, 2021.

[24] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *JASIST*, 60(5):911–932, 2009.

[25] 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.

[26] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[27] 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.

[28] 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.

[29] Purnamrita Sarkar, Sajid M Siddiqi, and Geogrey J Gordon. A latent space approach to dynamic embedding of co-occurrence data. In *AISTATS*, pages 420–427, 2007.

[30] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *ICONIP*, pages 362–373, 2018.

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *NeurIPS*, 27, 2014.

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

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[35] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Approximate graph propagation. In *KDD*, page 1686–1696, 2021.

[36] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *AAAI*, 2017.

[37] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *SIGMOD*, pages 2628–2638, 2021.

[38] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *ICLR*, 2021.

[39] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019.

[40] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *ICLR*, 2020.

[41] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 2018.

[42] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *KDD*, pages 2358–2366, 2022.

[43] Yuan Yuan, Xiaodan Liang, Xiaolong Wang, Dit-Yan Yeung, and Abhinav Gupta. Temporal dynamic graph lstm for action-driven video object detection. In *ICCV*, pages 1801–1810, 2017.

[44] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In *IJCAI*, pages 4419–4425, 2019.

[45] Yanping Zheng, Hanzhi Wang, Zhewei Wei, Jiajun Liu, and Sibo Wang. Instant graph neural networks for dynamic graphs. In *KDD*, pages 2605–2615, 2022.

[46] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. Tgl: A general framework for temporal gnn training on billion-scale graphs. In *VLDB*, page 1572–1580, 2022.

[47] Yuecai Zhu, Fuyuan Lyu, Chengming Hu, Xi Chen, and Xue Liu. Learnable encoder-decoder architecture for dynamic graph: A survey. *arXiv preprint arXiv:2203.10480*, 2022.