A survey of dynamic graph neural networks

Yanping ZHENG, Lu YI, Zhewei WEI

Gaoling School of Artificial Intelligence, Renmin University of China, Beijing 100872, China

Just Accepted

A survey of dynamic graph neural networks

Yanping ZHENG, Lu YI, Zhewei WEI(✉)

Gaoling School of Artificial Intelligence, Renmin University of China, Beijing 100872, China

Abstract Graph neural networks (GNNs) have emerged as a powerful tool for effectively mining and learning from graph-structured data, with applications spanning numerous domains. However, most research focuses on static graphs, neglecting the dynamic nature of real-world networks where topologies and attributes evolve over time. By integrating sequence modeling modules into traditional GNN architectures, dynamic GNNs aim to bridge this gap, capturing the inherent temporal dependencies of dynamic graphs for a more authentic depiction of complex networks. This paper provides a comprehensive review of the fundamental concepts, key techniques, and state-of-the-art dynamic GNN models. We present the mainstream dynamic GNN models in detail and categorize models based on how temporal information is incorporated. We also discuss large-scale dynamic GNNs and pre-training techniques. Although dynamic GNNs have shown superior performance, challenges remain in scalability, handling heterogeneous information, and lack of diverse graph datasets. The paper also discusses possible future directions, such as adaptive and memory-enhanced models, inductive learning, and theoretical analysis.

1 Introduction

With the remarkable advances in deep learning across various application domains, Graph Neural Networks (GNNs) have steadily emerged as a prominent solution for addressing problems involving complex graph-structured data. In the real world, crucial information is often represented in the form of graphs, such as relationships within social networks, roads and intersections in transportation systems, and protein interaction networks in bioinformatics. Compared to static graphs, dynamic graphs provide a more realistic representation of real-world systems and networks, as they can model structural and attribute information that evolves over time [1].

The nodes and edges of dynamic graphs may change over time, making traditional static GNNs difficult to apply directly, which introduces new challenges for the analysis and learning of dynamic graphs. Recently, researchers have integrated GNNs
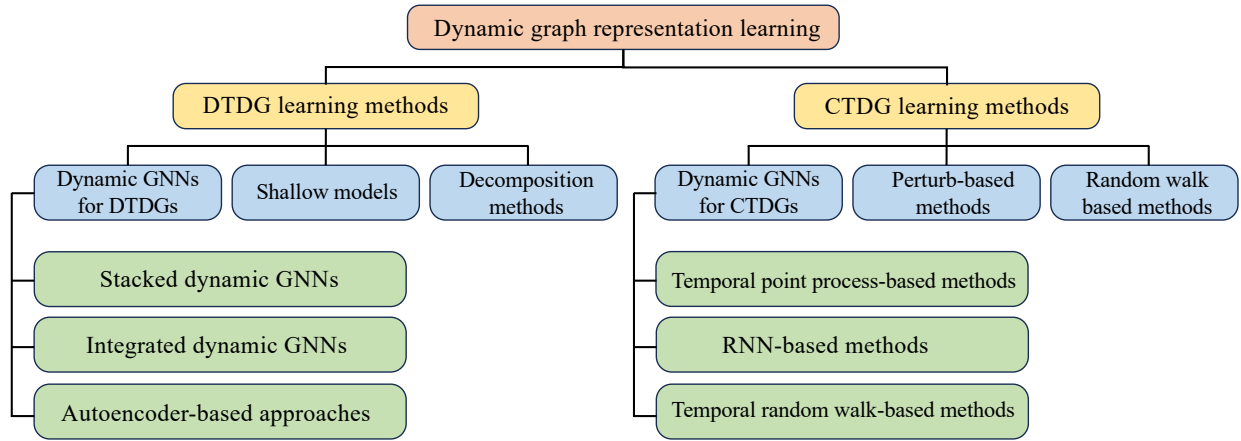
Figure 1   The taxonomy of dynamic graph representation learning.

with sequence learning to develop dynamic GNN models, such as the well-known TGAT [2], TGN [3] and ROLAND [4]. These models aggregate features from neighboring nodes and incorporate a time series module, enabling the modeling of both structural features and temporal dependencies within dynamic graphs. As a result, dynamic GNNs can generate evolving node representations that accurately capture the progression of time.

Although some work has surveyed methods for dynamic graph representation learning [1,5,6], with the continuous emergence of new methods and applications, the content of existing reviews has become outdated and cannot reflect the latest research developments and technological trends. Therefore, this paper offers a comprehensive review of recent developments in dynamic GNN models, encompassing representation and modeling techniques for dynamic graphs, along with a comparison of existing models. The taxonomy of existing methods is illustrated in Figure 1. We also explore new research directions, including large-scale applications and pre-training strategies, summarize the current state of research, and present an outlook on future trends.

Influenced by the available public datasets, node classification and link prediction are widely used as evaluation tasks for dynamic GNNs in current research. They are extensions of traditional static graph learning tasks to dynamic graphs, where features and labels may involve changes in the temporal dimension. Some studies also focus on domain-specific graph data to accomplish tasks such as knowledge graph completion [7], stock price prediction [8], and traffic flow prediction [9]. It is important to note that there exists a category of dynamic graphs in literature where the graph structure remains static, but the attributes of the nodes change over time. Such graphs are referred to as spatio-temporal graphs, which are considered to be outside of the scope of this survey. The main contributions are summarized as follows:

- We provide a comprehensive review of the fundamental concepts and characteristics of dynamic GNNs, construct the knowledge system of dynamic GNNs, and systematically investigate the key techniques and models.

- We present in detail the current mainstream dynamic GNN models and analyze in depth their advantages and disadvantages.

- We discuss the challenges facing this field, highlighting potential future research directions and technological trends.

The remainder of this paper is organized as follows. In Section 2 we begin with important definitions and background knowledge about dynamic graphs. The problem statement of dynamic graph representation learning is explicated in Section 3, which also provides a comprehensive review of dynamic GNNs following the taxonomy demonstrated in Figure 1. Section 4 focuses on models and systems designed for large-scale dynamic graphs. Section 5 summarizes the commonly used datasets, prediction tasks, and benchmarks. We find that new techniques such as transfer learning and pretrained technique are recently being applied to deal with the problem of dynamic graph learning, and we briefly review them in Section 6. We then discuss possible future directions in Section 7 and conclude the paper in Section 8.

2 Notation and Background

2.1 Notation

Table 1 summarizes the necessary notations used in this paper. We consider the undirected graph $G = (V, E)$, where V and E are the vertex and edge set, respectively. The node feature matrix is denoted as $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of nodes in the graph, and d is the dimension of the feature vector of each node. The connection between nodes in the graph is represented by the adjacency matrix $\mathbf{A}$. $\mathbf{D}$ is the diagonal degree matrix, and $\mathbf{P} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ denotes the normalized adjacency matrix, with $\mathbf{L} = \mathbf{I} - \mathbf{P}$ representing the Laplacian matrix. σ indicates the nonlinear activation function.

2.2 Dynamic Graphs

Dynamic graphs refer to intricate graph structures that involve temporal evolution. In these graphs, the nodes, edges, and properties within the graph

Table 1 Notations and the corresponding definitions.

Notation	Description
G	the graph
V	the vertex set
E	the edge set
n	the number of nodes
$\mathbf{X} \in \mathbb{R}^{n \times d}$	the feature matrix
$\boldsymbol{x}_i \in \mathbb{R}^d$	the feature vector of node i
$\mathbf{A} \in \mathbb{R}^{n \times n}$	the adjacent matrix
$\mathbf{D} \in \mathbb{R}^{n \times n}$	the degree matrix
$\mathbf{P} \in \mathbb{R}^{n \times n}$	the normalized adjacent matrix
$\mathbf{L} \in \mathbb{R}^{n \times n}$	the Laplacian matrix
σ	the nonlinear activation function
$\mathscr{G}$	the dynamic graph

demonstrate a state of continuous change. Dynamic graphs are commonly observed in various domains, including social networks [10], citation networks [11], biological networks [12], and so on. Taking social networks as an example, the registration of a new user results in the addition of new nodes to the graph, while the deactivation of existing users leads to a decrease in nodes. Moreover, when a user follows or unfollows another user, it represents the creation or disappearance of edges in the graph. Additionally, attributes like a user's age, location, and hobbies can change over time, indicating that node attributes are also dynamically evolving. Based on the granularity of time step, dynamic graphs can be categorized into Discrete-Time Dynamic Graphs (DTDGs) and Continuous-Time Dynamic Graphs (CTDGs).

Definition 1 (Discrete-Time Dynamic Graphs). Taking snapshots of the dynamic graph at equal intervals results in a discrete sequence of network evolution, defined as $\mathscr{G} = \{G^0, G^1, \cdots, G^T\}$, where $G^t = (V^t, E^t)$ $(0 \le t \le T)$ is the t-th snapshot.

As shown in Figure 2(c), DTDGs decompose the dynamic graph into a series of static networks, capturing the graph structure at various selected
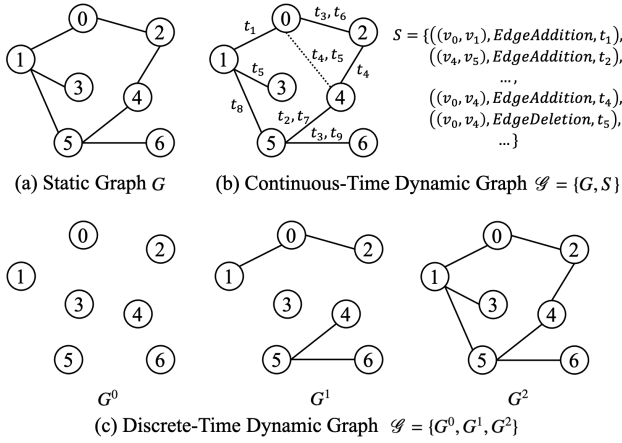
(a) Static Graph G (b) Continuous-Time Dynamic Graph $\mathscr{G} = \{G, S\}$

$$S = \{((v_0, v_1), EdgeAddition, t_1),$$
$$((v_4, v_5), EdgeAddition, t_2),$$
$$\cdots,$$
$$((v_0, v_4), EdgeAddition, t_4),$$
$$((v_0, v_4), EdgeDeletion, t_5),$$
$$\cdots\}$$

G^0 G^1 G^2

(c) Discrete-Time Dynamic Graph $\mathscr{G} = \{G^0, G^1, G^2\}$

Figure 2 Comparative illustration of different graph representations. (a) Representation of a static graph where the structure remains unchanged over time, without temporal information; (b) Illustration of a Continuous-Time Dynamic Graph (CTDG) where interactions between nodes are labeled by timestamps, and multiple interactions are allowed; (c) Representation of a Discrete-Time Dynamic Graph (DTDG), which captures the evolution of relationships in discrete time intervals.

time points. However, some intricate details might be lost within the chosen intervals. Moreover, determining time intervals that balance accuracy and efficiency presents a significant challenge.

Definition 2 (Continuous-Time Dynamic Graphs). A CTDG is typically defined as $\mathscr{G} = \{G, S\}$, where G denotes the initial state of the graph at time t_0. This initial state might be empty or include initial structures (associated edges) but does not contain any history of graph events. The set $S = \{event^1, event^2, \cdots, event^T\}$ represents a collection of graph events. Each graph event is defined by a triple, which consists of the nodes participating in the event, its associated event type, and a timestamp. The event type signifies the various possible modifications occurring within a graph, including the addition or deletion of edges or nodes, as well as updates to node features. Fundamentally, these modifications may result in changes to the structure or attributes of the graph.

As shown in Figure 2(b), the initial graph G is gradually updated according to the guidance of S. The structure of the graph at time $t = t_9$ is consistent with Figure 2(a), and the complete process of change is recorded in S. It should be noted that an edge previously existed between node v_0 and node v_4, which quickly disappeared. However, the DTDG illustrated in Figure 2(c) does not capture this dynamic.

Remark. To ensure clarity and consistency in this paper, unless otherwise specified, the graph at time t refers to the t-th version of the graph. This could mean either the t-th graph snapshot in a DTDG or a graph updated with events up to time t from its initial state in a CTDG.

Definition 3 (Temporal Neighbors). At time t, the temporal neighbor of node v is represented as $N^t(v)$. Based on the traditional neighbor set, each neighbor node w in the set is associated with a timestamp t' to form a $(node, timestamp)$ pair. It indicates that at time t', node v is connected to node w. It is required that the timestamp t' corresponding to a neighboring node is smaller than the current timestamp t. In other words, $N^t(v)$ represents the collection of adjacent nodes resulting from events occurring prior to time t.

2.3 Hawkes Processes

Introduced by Hawkes in 1971 [13], the Hawkes process is a mathematical model for time-sequenced events. Based on the assumption that past events influence current events, the Hawkes process simulates a point process with an underlying self-exciting pattern. It is appropriate for modeling the evolution of discrete sequences. Given all historical events up to time t, the probability of an event occurring in a short time interval $[t, t + \Delta)$ at time t can be defined by the conditional intensity function of the Hawkes

process, denoted as

$$\lambda^*(t) = \lim_{\Delta t \to 0} \frac{\mathbb{E}[H(t + \Delta t) - H(t) \mid \mathcal{H}(t)]}{\Delta t}$$
$$= \mu + \int_0^t g(t - s)dN(s), \tag{1}$$

where $\mathcal{H}(t)$ represents historical events prior to time t, $H(t)$ denotes the number of historical events that occurred before time t, and Δt indicates the window size. The first equation describes the probability that an event will occur in the region beyond the observed data $\mathcal{H}(t)$. Specifically, it represents the probability that an event will occur at time $t + \Delta t$ based on prior observations up to time t. The background intensity is represented by μ, and the excitation function is denoted by $g(\cdot)$. Typically, $g(\cdot)$ is defined as an exponential decay function, such as $e^{-\beta t}(\beta > 0)$, which describes the phenomenon where the influence of an event diminishes as the time interval increases. According to the second equation, $\lambda^*(\cdot)$ is a flexible function that depends only on the background intensity and the excitation function. Therefore, for an observed event sequence $\{t_1, t_2, t_3, \cdots\}$, the intensity function of the Hawkes process is defined as

$$\lambda^*(t) = \mu + \sum_{t_i < t} \alpha_i g(t - t_i), \tag{2}$$

where α_i represents the weight of the impact of each event at time t.

Initially, the Hawkes process was utilized for sequence modeling [14, 15]. Following the development of graph representation learning, HTNE [16] first introduced the Hawkes process into dynamic graph representation learning. It models the evolutionary incentive of historical behaviors on current actions, thereby simulating the graph generation process. Lu et al. [17] proposed a dynamic graph representation model that combines macroscopic graph evolution with microscopic node edge predic-

tion, thereby capturing the dynamic nature of the graph structure and the rising trend of graph scale.

3 Graph Representation Learning

The objective of graph representation learning is to build low-dimensional vector representations of nodes while preserving important information such as structural, evolutionary, or semantic features. This has emerged as a crucial concern in graph data mining and analysis in recent years. The learned representations have extensive applications in various tasks, including node classification, link prediction, and recommendation.

It serves as a bridge between raw graph data and graph analysis tasks by encoding each node into a vector space. This approach transforms graph learning tasks into corresponding problems in machine learning applications. For example, we can effectively and efficiently address these graph analysis challenges by using the learned representations as the input feature for traditional machine learning algorithms. Additionally, we can further enhance this process by leveraging suitable classification and clustering techniques for tasks such as node classification and community detection.

Definition 4 (Graph Representation Learning). Given a graph $G = (V, E)$, the task of graph representation learning is to learn an efficient mapping function $\mathcal{F}(v) \to \mathbf{z}_v$, where $\mathbf{z}_v \in \mathbb{R}^d$ represents the low-dimensional representation of node v, and $d \ll n$, n is the number of nodes in the graph.

Graph representation learning has been extensively studied in recent years. Earlier methods attempted to decompose data matrices into lower-dimensional forms while preserving the manifold structures and topological properties inherent to the original matrices. Prominent examples of this

approach include techniques such as Local Linear Embedding (LLE) [18] and Laplacian Eigenmaps (LE) [19]. These matrix factorization-based methods were further developed to incorporate higher-order adjacency matrices, thereby preserving the structural information of graphs, such as GraRep [20] and HOPE [21]. Another strategy for graph embedding creates multiple paths by traversing from random initial nodes to capture contextual information about global and local structural information of neighboring nodes, namely random walk-based methods. Probabilistic models such as Skip-gram [22] and Bag-of-Words [23] are then employed to sample these paths and learn node representations randomly, and typical examples include DeepWalk [24] and Node2vec [25]. The success of deep learning techniques has paved the way for an abundance of deep neural network-based graph representation learning techniques. Graph embedding methods based on deep learning typically apply deep neural networks directly to the entire graph (or its adjacency matrix), encoding it into a lower-dimensional space. In this field, notable methods include SDNE [26], DNGR [27], ChebNet [28], GCN [29], GAT [30], and GraphSAGE [31].

Unlike traditional static graphs, which focus solely on maintaining structural features, dynamic graphs also incorporate evolutionary features. Dynamic graph representation learning employs graph representation techniques for dynamic graphs and tracks the evolution of node representations. As the graph evolves over time, the representations of the nodes adapt accordingly. A time-dependent function expresses this evolution, and we denote the updated representation vector following an event involving node v at time t as $\mathbf{z}_v^t$.

Definition 5 (Dynamic Graph Representation Learning). Given a dynamic graph $\mathscr{G}$ evolving over time,

the task of dynamic graph representation learning is to learn an efficient mapping function $\mathcal{F}(v, t) \rightarrow \mathbf{z}_v^t$, where $\mathbf{z}_v^t \in \mathbb{R}^d$ represents the time-dependent low-dimensional representation of node v at time t. Here, $d \ll n^t$, and n^t is the number of nodes in the graph at time t.

3.1 Models for Dynamic graphs

Corresponding to the different types of dynamic graphs described in Definitions 1 and 2, dynamic graph representation learning models can be divided into two main categories: discrete-time dynamic graph representation learning and continuous-time dynamic graph representation learning. The former divides the dynamic graph into multiple snapshots, which can be regarded as multiple static graphs. Consequently, methods designed for static graphs can be used to learn representations for each snapshot, while also analyzing interrelationships and evolutionary trends between successive snapshots. In continuous-time dynamics, interactions between graph nodes are tagged with timestamps, effectively differentiating each edge based on its timestamp. Continuous-time dynamic graph representation learning refers to the methods that utilize this edge information with accompanying timestamps for learning representations.

By introducing matrix perturbation theory, the matrix decomposition-based representation learning method designed for static graphs can be extended to the learning of dynamic graphs. A few assumptions and operations form the basis for matrix perturbation theory. First, it is assumed that the adjacency matrix of the current time step is a perturbed version of the matrix from the previous time step. Then, a common low-rank decomposition is performed on these two matrices, ensuring that the representation matrix of the current time step is updated smoothly

from the previous time step. By constraining the magnitude of the perturbation, it is possible to regulate the smoothness of the representation between time steps [32, 33]. However, this method is more suited to dealing with DTDGs than CTDGs, as it may face challenges such as high computational complexity, difficulties in real-time updates, and inadequacies in capturing temporal dependencies on frequently changing CTDGs.

By requiring that random walks on graphs strictly follow the order of timestamps, it is possible to perform temporal walks on dynamic graphs, particularly suitable for CTDGs. For example, CTDNE [34] incorporates a biased sampling mechanism that exhibits temporal information during the sampling process, ensuring a higher probability of sampling the edges with smaller time intervals. EvoNRL [35], on the other hand, designs strategies for four distinct scenarios in dynamic graphs, i.e., the addition and removal of edges or nodes, updating node representations with each change of the graph.

Traditional methods rely more frequently on manually crafted features and customized models, resulting in limited learning and adaptability. Rooted in deep learning principles, GNNs have shown a remarkable ability to learn complex patterns and features in static graphs, exhibiting improved representation learning performance as well as superior generalization abilities. As a result, there has been a growing interest in utilizing GNN-inspired methods in the field of dynamic graph learning. These approaches leverage the powerful learning capabilities of GNNs to enhance the comprehension of dynamic graphs that exhibit complex temporal evolution patterns, which are known as Dynamic Graph Neural Networks (Dynamic GNNs). Recently, there has been an explosion of work on the design of Dynamic GNNs. Therefore, we introduce

Dynamic GNNs that deal with DTDGs and CTDGs in Sections 3.2 and 3.3, respectively. These two types of methods capture temporal evolution patterns in various dynamic graphs and have achieved excellent performance in several dynamic graph analysis tasks. To highlight the strengths and weaknesses of each method, we provide an in-depth comparison in Table 2, focusing on the base models used, the types of graphs supported, and the graph events handled.

3.2 Dynamic GNNs for DTDGs

DTDGs are composed of multiple snapshots that are chronologically organized and can be modeled as sequential data, as described in Definition 1. Therefore, temporal patterns in DTDGs are measured by the sequential relationships between different snapshots. The prevailing research methodology involves employing static graph neural network models to represent each graph snapshot individually. These snapshots are subsequently organized chronologically and treated as sequential data, which can be input into sequence models to comprehensively determine the interplay between different graph snapshots and learn temporal patterns. As a representative sequence model, Recurrent Neural Networks (RNNs) are often combined with GNNs to form the mainstream dynamic GNNs for DTDGs. Based on how they are combined, they can be broadly classified as stacked architectures or integrated architectures.

Stacked dynamic GNNs combine spatial GNNs with temporal RNNs in a modular fashion to model discrete sequences of dynamic graphs. As shown in Figure 3(a), the distinct components are dedicated to spatial and temporal modeling, respectively. Specifically, these models use distinct GNNs to aggregate spatial information on each graph snapshot individ-
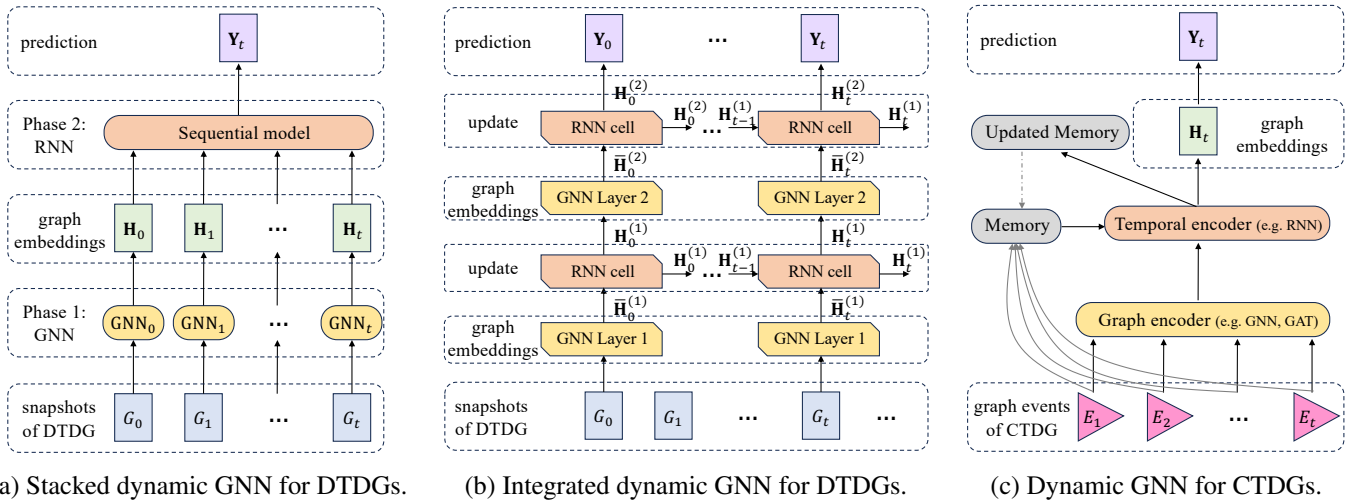
(a) Stacked dynamic GNN for DTDGs. (b) Integrated dynamic GNN for DTDGs. (c) Dynamic GNN for CTDGs.

Figure 3   Different model architectures for dynamic graphs.

ually. The outputs from the different time steps are then fed into a temporal module for sequence modeling. Therefore, given a graph G^t at time t, the representation is computed as follows:

$$\mathbf{Z}^t = f\left(G^t\right), \mathbf{H}^t = g\left(\mathbf{H}^{t-1}, \mathbf{Z}^t\right), \qquad (3)$$

where f is a GNN model used to encode G^t and obtain the graph representation $\mathbf{Z}^t$ at this time step. g is a sequence model such as RNN, used to integrate the previous hidden state $\mathbf{H}^{t-1}$ and current graph representation $\mathbf{Z}^t$ to update the current hidden state $\mathbf{H}^t$. Manessi et al. [36] use this architecture to propose CD-GCN and WD-GCN models, where f is a GCN model and g uses a standard Long Short-Term Memory (LSTM). Notably, each node utilizes a distinct LSTM, though the weights can be shared. The difference between WD-GCN and CD-GCN is that CD-GCN adds skip connections to GCN.

RNNs are not the only available option for learning time series. In recent studies, GNNs have also been used in combination with other kinds of deep time series models. For instance, DySAT [37] introduces a stacked architecture made up entirely of self-attention blocks. They use attention mechanisms in both the spatial and temporal dimensions, employing Graph Attention Networks (GAT) [30]

for the spatial dimension and Transformers [14] for the temporal dimension. Similarly, models like TNDCN [38] and TEDIC [39] combine GNNs with one-dimensional spatio-temporal convolutions.

Integrated dynamic GNNs merge GNNs and RNNs within a single network layer, thus amalgamating spatial and temporal modeling within the same architectural level, as shown in Figure 3(b). GC-LSTM [40] uses the adjacency matrix of a given time as input to the LSTM and performs spectral graph convolution [61] on the hidden layer. The model takes a sequence of adjacency matrices as input and returns node representations that encode both temporal and spatial information. Within this framework, the GNN learns the topological features of the LSTM's cell and hidden states, which are used to maintain long-term relationships and extract input information, respectively. This implies that the node representations provided by the GNN are distinct from the node history captured by the LSTM. Let $f_{\mathbf{i}}$, $f_{\mathbf{f}}$, $f_{\mathbf{u}}$, $f_{\mathbf{o}}$, and $f_{\mathbf{c}}$ represent the five GNN models acting on the input gate, forget gate, update gate, output gate, and cell state, respectively. They possess the same structure but different parameters. The parameters of $f_{\mathbf{i}}$, $f_{\mathbf{f}}$, $f_{\mathbf{u}}$, and $f_{\mathbf{o}}$ are initialized by the hidden state $\mathbf{H}^{t-1}$ from the

Table 2 Comparison of graph neural network methods in literature review. ✓ represents that the method supports the corresponding graph event, ✗ means it does not support that event, and □* denotes that it considers only static attributes.

Method	Models	Graph	Insertion		Deletion		Attribute	
			Node	Edge	Node	Edge	Node	Edge
WD-GCN [36]	GCN with skip connections and LSTM	DTDG	✗	✓	✗	✓	✓	✓
DySAT [37]	GAT and Transformer	DTDG	✓	✓	✓	✓	✓	✓
TNDCN [38]	GNNs and spatio-temporal convolutions	DTDG	✗	✓	✗	✓	✓	✓
TEDIC [39]	GNNs and spatio-temporal convolutions	DTDG	✗	✓	✗	✓	✓	✓
GC-LSTM [40]	GCN and LSTM	DTDG	✗	✓	✗	✓	✓	✓
LRGCN [41]	R-GCN and LSTM	DTDG	✗	✓	✗	✓	✓	✓
RE-Net [42]	integrate R-GCN within RNNs	DTDG	✗	✓	✗	✓	✗	✗
WinGNN [43]	GNN and randomized sliding-window	DTDG	✓	✓	✓	✓	□*	✗
ROLAND [4]	GNNs and RNNs	DTDG	✓	✓	✓	✓	✓	✓
EvolveGCN [44]	GCN and LSTM/GRU	DTDG	✓	✓	✓	✓	✓	✓
SEIGN [45]	GCN and LSTM/GRU	DTDG	✓	✓	✓	✓	✓	✓
dyngraph2vec [46]	AE and LSTM	DTDG	✓	✓	✓	✓	✗	✗
Know-Evolve [47]	RNN-parameterized TPP	CTDG	✗	✓	✗	✗	✗	✗
DyREP [48]	RNN-parameterized TPP	CTDG	✓	✓	✗	✗	✗	✗
LDG [49]	RNN-parameterized TPP and self attention	CTDG	✓	✓	✗	✗	✗	✗
GHNN [50]	LSTM-parameterized TPP	CTDG	✗	✓	✗	✗	✗	✗
GHT [51]	LSTM-parameterized TPP	CTDG	✗	✓	✗	✗	✗	✗
TREND [52]	Hawkes process-based GNN	CTDG	✓	✓	✗	✗	□*	✗
DGNN [53]	GNNs and LSTM	CTDG	✓	✓	✗	✗	✗	✗
JODIE [54]	GNNs and RNNs	CTDG	✓	✓	✗	✗	□*	✓
TGAT [2]	GAT, time encode and RNNs	CTDG	✓	✓	✗	✗	□*	✓
TGN [3]	GNNs, time encode and RNNs	CTDG	✓	✓	✗	✗	□*	✓
APAN [55]	GNNs and RNNs	CTDG	✓	✓	✗	✗	□*	✓
CAW-N [56]	causal anonymous walk based GNN	CTDG	✓	✓	✗	✗	✗	✓
Zebra [57]	temporal PPR based GNN	CTDG	✓	✓	✗	✗	✓	✓
EARLY [58]	GCN and temporal random walk	CTDG	✓	✓	✓	✓	✓	✓
Zheng et al. [59]	decoupled GNNs and sequential models	CTDG	✓	✓	✓	✓	✓	✓
SDG [60]	APPNP and dynamic propagation	CTDG	✗	✓	✗	✓	✓	✓

previous time step, while $f_{\mathbf{c}}$ is initialized based on the memory $\mathbf{C}^{t-1}$:

$$\mathbf{i}^t = \sigma\left(\mathbf{A}^t\mathbf{W_i} + f_{\mathbf{i}}\left(G^{t-1}\right) + \mathbf{b_i}\right)$$
$$\mathbf{f}^t = \sigma\left(\mathbf{A}^t\mathbf{W_f} + f_{\mathbf{f}}\left(G^{t-1}\right) + \mathbf{b_f}\right)$$
$$\mathbf{C}^t = \mathbf{f}^t \otimes f_{\mathbf{c}}\left(G^{t-1}\right) + \mathbf{i}^t \odot \tanh\left(\mathbf{A}^t\mathbf{W_c} + f_{\mathbf{u}}\left(G^{t-1}\right) + \mathbf{b_c}\right)$$
$$\boldsymbol{o}^t = \sigma\left(\mathbf{A}^t\mathbf{W_o} + f_{\mathbf{o}}\left(G^{t-1}\right) + \mathbf{b_o}\right)$$
$$\mathbf{H}^t = \boldsymbol{o}^t \odot \tanh\left(\mathbf{C}^t\right),$$

(4)

where $\{\mathbf{W_i}, \mathbf{W_f}, \mathbf{W_o}, \mathbf{W_c}\}$ and $\{\mathbf{b_i}, \mathbf{b_f}, \mathbf{b_c}, \mathbf{b_o}\}$ are the weight and bias matrices, respectively, and $\mathbf{A}^t$ is the adjacency matrix of G^t. Other integrated approaches often follow a similar framework, differing perhaps in the GNN or RNN models they employ, the use cases, or the types of graphs they focus on. The Long Short-Term Memory R-GCN (LRGCN) [41] adopts a similar strategy, incorporating topological variations directly into its computations. Specifically, LRGCN [41] integrates a Relational GCN (R-GCN) [41] within the LSTM framework. The

computations for the input gate, forget gate and output gate are the results of the R-GCN model acting on the input node representations and embeddings computed from the previous step. Meanwhile, RE-Net [42] integrates R-GCN [41] into multiple RNNs, enabling it to learn from dynamic knowledge graphs. WinGNN [43] utilizes the idea of randomized gradient aggregation over sliding windows to model temporal patterns without extra-temporal encoders. Specifically, WinGNN introduces a stochastic gradient aggregation mechanism based on sliding windows, gathering gradients across multiple consecutive snapshots to attain a global optimum. In addition, an adaptive gradient aggregation strategy is designed to assign different weights to gradients from different snapshots, overcoming the influence of local optima. It also introduces a snapshot random dropout mechanism to enhance the robustness of gradient aggregation further.

ROLAND [4] extends the GNN-RNN framework by utilizing hierarchical node states. Specifically, You et al. [4] propose to stack multiple GNN layers and interleave them with a sequence model such as RNN, where the node embedding of different GNN layers is regarded as hierarchical node states that are continuously updated over time. In contrast to traditional GNNs, ROLAND uses not only the embedding information $\mathbf{H}_t^{(\ell-1)}$ from the previous layer but also the information $\mathbf{H}_{t-1}^{(\ell)}$ from the previous moment of the current layer, when updating the embedding matrix $\mathbf{H}_t^{(\ell)}$ at time t. Thus, the (ℓ)-th layer of the framework is

$$\overline{\mathbf{H}}_t^{(\ell)} = f^{(\ell)}\left(\overline{\mathbf{H}}_t^{(\ell-1)}\right), \mathbf{H}_t^{(\ell)} = g\left(\mathbf{H}_{t-1}^{(\ell)}, \overline{\mathbf{H}}_t^{(\ell)}\right), \quad (5)$$

where $f^{(\ell)}$ is the (ℓ)-th GNN layer, g is the sequence encoder. Bonner et al. [62] propose stacking a GCN layer, a Gated Recurrent Unit (GRU) layer [63], and a linear layer to form a Temporal Neighborhood Aggregation (TNA) layer, and employ a two-layer model to implement 2-hop convolution combined with variational sampling for link prediction.

Instead of updating node representations with RNNs as in previous methods, EvolveGCN [44] uses RNNs to update the weight parameters of GCNs between graph snapshots, facilitating dynamic adaptation and model performance improvement. Building on this, they introduced two variants: EvolveGCN-H, employing a GRU to enhance temporal stability, and EvolveGCN-O, using an LSTM to capture longer dependencies. To improve scalability, SEIGN [45] focuses on evolving parameters associated with the graph filter, specifically the convolutional kernel, rather than the entire GNN model.

Autoencoder-based approaches are trained distinctively compared to the dynamic graph neural network models discussed above. They leverage dynamic graph embeddings to capture both the topology and its temporal changes. DynGEM [64], for instance, employs an autoencoder to learn the embeddings from each snapshot of the DTDG. Interestingly, the autoencoder at time t initializes its parameters based on the autoencoder from time $t-1$. The architecture of this autoencoder, including the number of layers and neurons in each layer, is adjusted based on the differences between the current snapshot and its predecessor. The Net2Net [65] technique is utilized to maintain the mapping relationships across layers when modifications are made to the autoencoder. To effectively capture the relationships and changes within dynamic graphs, dyngraph2vec [46] employs a series of adjacency matrices $\{\mathbf{A}_{t-w+1}, \cdots, \mathbf{A}_{t-1}\}$ from within a time window as input, where w represents the size of the time window. An autoencoder is then utilized to obtain the adjacency matrix $\mathbf{A}_t$ for the current timestep t.

3.3 Dynamic GNNs for CTDGs

As defined in Definition 2, CTDGs are not stored as snapshots with fixed intervals, but each event is recorded as a triplet (node, event type, timestamp), e.g., $((u, v), EdgeAddition, t)$ denotes an edge insertion event between nodes u and v at time t. Therefore, each event in a continuous dynamic graph has a timestamp, which smoothly tracks the evolution of the graph structure and allows for more accurate analysis of temporal relations and dependencies between events. In addition, discrete graphs synchronously aggregate time-slice information and cannot distinguish the sequence of events, whereas continuous graphs specify the temporal order of events and support modeling of asynchronous time intervals. Approaches to learning continuous dynamic graphs revolve around how to effectively capture and characterize the evolution of graph structure over time, as shown in Figure 3(c). There are currently three classes of approaches to continuous-type dynamic graph neural networks: (1) temporal point process-based approaches, in which the temporal point process is parameterized by a neural network; (2) RNN-based approaches, in which node representations are maintained by an RNN-based architecture; and (3) temporal random walk-based approaches, whose essence is to capture both temporal and structural properties of graphs through temporal random walks.

Temporal point process-based methods. Know-Evolve [47] models knowledge graphs as interactive networks, where temporal point processes are used to model the occurrence of events. It defines a bilinear relationship function for representing multi-relational interactions between nodes. The central tenet of the method is to model the occurrence of facts as a multivariate temporal point process,

whose conditional intensity function is modulated by the score of the relevant relationship. This relationship score is derived from embeddings of nodes that evolve dynamically. In addition, two custom RNNs are used to update node embeddings as new graph events occur.

Based on a similar strategy, Trivedi et al. proposed DyRep [48], a method for modeling multiple types of graphs. DyRep develops a pairwise intensity function that quantifies the strength of relationships between nodes. It further employs self-attention mechanisms to construct a temporal point process, which enables the model to prioritize relevant information for the current task, filtering out irrelevant details. Therefore, the structural embedding for node u is defined as: $\mathbf{h}_{\text{struct}}^{\bar{t}}(u) = \max\left(\left\{\sigma\left(q^t(u, i) \cdot \mathbf{h}^{\bar{t}}(i)\right), \forall i \in N^{\bar{t}}(u)\right\}\right)$, where $\mathbf{h}^{\bar{t}}(i) = \mathbf{W}_h \mathbf{z}^{\bar{t}}(i) + \mathbf{b}_h$, $\mathbf{z}^{\bar{t}}(i)$ is the embedding of the most recent node i, $\bar{t}$ represents the most recent time node i was active, and $q(u, i)$ denotes the aggregation weight corresponding to the neighbor node i. Specifically, the model computes the relationship between node u and its neighbor node $i \in N^t(u)$ by

$$q^t(u, i) = \frac{\exp\left(\mathbf{S}^t(u, i)\right)}{\sum_{i' \in N^t(u)} \exp\left(\mathbf{S}^{\bar{t}}(u, i')\right)}, \qquad (6)$$

where $\mathbf{S}^t \in \mathbb{R}^{n \times n}$ is the attention matrix, which represents the fraction of attention between pairs of nodes at time t. $\mathbf{S}$ is computed and maintained by the adjacency matrix $\mathbf{A}$ and the conditional intensity function, and the conditional intensity function between node u and node v is defined as

$$\lambda^t(u, v) = f\left(g^{\bar{t}}(u, v)\right), \qquad (7)$$

where $g^{\bar{t}(u,v)}$ is an internal function that computes the compatibility between two recently updated nodes: $g^{\bar{t}}(u, v) = w^T \cdot \left[\mathbf{z}^{\bar{t}}(u) : \mathbf{z}^{\bar{t}}(v)\right]$, and : denotes the concatenation, w^T is the learnable scale-specific

compatibility model parameter. $f(\cdot)$ uses the modified softplus function $f(x) = \psi \log\left(1 + \exp\left(\frac{x}{\psi}\right)\right)$, where x represents $g^t(\cdot)$, $\psi > 0$ is the learnable scalar time parameter corresponding to the proportion of relevant events occurring during the process.

Latent Dynamic Graph (LDG) [49] is an enhanced version of DyREP [48] that integrates Neural Relational Inference (NRI) [66] to encode temporal interactions, thereby optimizing the attention mechanism and resulting in improved performance. Graph Hawkes Neural Network (GHNN) [50] and Graph Hawkes Transformer (GHT) [51] model temporal dependencies between nodes using Hawkes processes, and estimate the intensity functions using continuous-time LSTM and Transformer respectively to learn time-evolving node representations. TREND [52] also designs a Hawkes process-based GNN to learn representations for CTDGs, where the parameters of the conditional intensity are learned through a Fully Connected Layer (FCL), and Feature-wise Linear Modulation (FiLM) is used to learn the parameters of the FCL.

RNN-based methods. These models use RNNs to maintain node embeddings in a continuous manner, which allows them to respond to graph changes in real time and update the representations of affected nodes. Specifically, such models define an RNN unit to track the evolution of each node embedding. When the graph changes, the RNN unit can calculate the new hidden state at the current time t based on the current input and the hidden state at the previous moment. Therefore, one of the main differences between these methods lies in how they define the embedding function and customize the RNN.

DGNN [53] consists of an update component and a propagation component. When introducing a new edge, the update component maintains the freshness of node information by capturing the order of edge additions and the time interval between interactions, implemented by the LSTM model. The propagation component propagates the new interaction information to the affected nodes by considering the intensity of the influence.

JODIE [54] is composed of two operations: the update operation and the projection operation. The former is used to update the embedding information of nodes, while the latter is used to predict future embedding trajectories. Moreover, each node has a static embedding to represent its fixed attributes, and a dynamic embedding to reflect its current state. JODIE focuses on interaction network data. In the update operation, the embeddings of the user node u and the item node i are updated by two RNNs that have the same structure but different parameters:

$$\mathbf{z}^t(u) = \sigma\left(\mathbf{W}_{u,1}\mathbf{z}^{\bar{t}}(u) + \mathbf{W}_{u,2}\mathbf{z}^{\bar{t}}(i) + \mathbf{W}_{u,3}e_{u,i} + \mathbf{W}_{u,4}\Delta_u\right)$$
$$\mathbf{z}^t(i) = \sigma\left(\mathbf{W}_{i,5}\mathbf{z}^{\bar{t}}(i) + \mathbf{W}_{i,6}\mathbf{z}^{\bar{t}}(u) + \mathbf{W}_{i,7}e_{u,i} + \mathbf{W}_{i,8}\Delta_i\right),$$
$$(8)$$

where $\mathbf{z}^{\bar{t}}(u)$ and $\mathbf{z}^{\bar{t}}(i)$ represent the previous embeddings of node u and node i respectively. Δ_u and Δ_i represent the time difference from the last interaction. $e_{u,i}$ is the feature vector corresponding to the current interaction between node u and node i. The sets $\{\mathbf{W}_{u,1}, \cdots, \mathbf{W}_{u,4}\}$ and $\{\mathbf{W}_{i,5}, \cdots, \mathbf{W}_{i,8}\}$ represent the parameters for the two RNN, RNN_u and RNN_i, respectively. σ is the sigmoid function.

The embedding projection operation is designed to generate the future embedding trajectory of a user, and this embedding can be applied to downstream tasks. The projection operation takes two inputs: (1) the embedding of node u at time t, denoted as $\mathbf{z}^t(u)$, and (2) the time difference, Δ. By projecting $\mathbf{z}^t(u)$, the representation of node u over a certain time interval can be learned as:

$$\hat{\mathbf{z}}^{t+\Delta}(u) = \left(1 + \mathbf{W}_p\Delta\right) * \mathbf{z}^t(u), \qquad (9)$$

where $\mathbf{W}_p$ is the learned matrix. As Δ increases, the offset of the projection embedding becomes larger. Assume that item j interacts with user u at time $t+\Delta$, the embedding of node j at time $t + \Delta$ is predicted based on a fully connected linear layer:

$$\mathbf{z}^{t+\Delta}(j) = \mathbf{W}_1\hat{\mathbf{z}}^{t+\Delta}(u) + \mathbf{W}_2\bar{\mathbf{z}}(u) + \mathbf{W}_3\mathbf{z}^{t+\Delta}(i) + \mathbf{W}_4\bar{\mathbf{z}}(i) + \mathbf{B},$$
(10)

where $\bar{\mathbf{z}}(u)$ and $\bar{\mathbf{z}}(i)$ represent the static embeddings of nodes u and i, respectively. $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4$ and $\mathbf{B}$ are the parameters of this linear layer.

Inspired by Time2Vec [67], TGAT [2] introduces a time encoding function based on the Bochner theorem [68], which can map continuous time to a vector space. This method presents a viable substitute for position encoding within self-attention mechanisms, effectively leveraging temporal information. By utilizing the improved self-attention framework, TGAT [2] presents a temporal graph attention layer specifically designed to incorporate feature data from historical neighbors of a target node.

TGN [3] presents a unified approach by integrating the methodological frameworks of TGAT [2], JODIE [54], and DyREP [48], which is composed of five core modules: Memory, Message Function, Message Aggregator, Memory Updater, and Embedding. The Memory module stores historical information about nodes, and the Embedding module produces real-time representations. Whenever an event involving a node occurs, the corresponding memory is updated. This update function can be learning models like RNNs or non-learning methods like averaging or aggregation, and thus take advantage of previous messages to refresh the memory during training, facilitating the generation of node embeddings. Furthermore, by ingeniously architecting each module, TGN can encapsulate a variety of models, including the likes of JODIE [54],

DyRep [48], and TGAT [2].

During online training, it cannot be guaranteed that graph update events arrive in timestamp order, which will make RNN-based embedding generation unstable. APAN [55] addresses this issue by employing an asynchronous propagation mechanism. It allows messages to propagate through the graph asynchronously by specifying the number of propagation steps and thus ensures events are stored in order of their timestamps.

Temporal random walk-based methods. Inspired by models such as DeepWalk [24] and Node2vec [25], Nguyen et al. [34] first proposed a temporal random walk based on the properties of dynamic graphs, where a random walk consists of a sequence of edges with non-decreasing timestamps. Therefore, the movement from one vertex to another is dictated not only by the adjacency of the nodes but also by temporal constraints, ensuring that the walks align with both the graph's structure and the sequence of time.

Starting from links of interest, Causal Anonymous Walk (CAW) [56] traces back several neighboring links to encode the latent causal relationships in network dynamics. During walks, CAW [56] removes node identities to facilitate inductive learning and instead encodes relative node identities based on their frequencies in a set of sampled walks. This method of relative node identification ensures that pattern structures and their correlations are preserved after node identities are removed, achieving set-based anonymization. To predict temporal links between two nodes, the CAW-Network (CAW-N) model is proposed. It samples some CAWs associated with the two nodes, encodes and aggregates these CAWs using RNNs and set pooling, respectively, to make predictions.

Zebra [57] introduces the Temporal Personalized

PageRank (T-PPR) metric that uses an exponential decay model to estimate the influence of nodes in dynamic graphs. Given a source node (i, τ_1) and a target node (j, τ_2), where (i, τ_1) represents node i at time τ_1, the T-PPR value $\pi_{i,\tau_1}(j, \tau_2)$ of the target node (j, τ_2) with respect to the source node (i, τ_1) is defined as the probability of starting a temporal random walk from (i, τ_1) with a parameter α $(0 < \alpha < 1)$, and eventually stopping at (j, τ_2). T-PPR operates under the premise that recent neighboring nodes hold paramount importance. Therefore, by applying an exponential decay model parameterized by β $(0 < \beta < 1)$, it ensures that the probability of traversing the most distant ones is the lowest among all the historical neighbors of a given node. For example, suppose source node (v_1, t_6) has three historical neighbor nodes (v_4, t_2), (v_2, t_4) and (v_3, t_5), assigned with transition probabilities β^3, β^2, β. According to the definition, the T-PPR value of (v_3, t_5) can be computed as: $\pi_{v_1,t_6}(v_3, t_5) = \frac{\beta}{\beta+\beta^2+\beta^3} \times \alpha(1-\alpha)$. Based on the T-PPR values, Zebra [57] can discern the top-k temporal neighbors with the most significant impact on the target node. It directly aggregates these k high-influence temporal neighbor nodes to generate the target node's embedding, avoiding complex recursive temporal neighbor aggregation.

4 Processing and Learning on Large-scale Dynamic Graphs

Several strategies are available for managing large-scale static graphs. First, to improve processing efficiency, distributed parallel training can be implemented by dividing the graph across multiple devices for simultaneous processing, which ensures data integrity. Second, to enhance throughput and optimize resource use, techniques such as pipeline parallelism, data parallelism, and model parallelism are widely employed. Third, to address computational complexity, sampling methods and neighbor truncation are utilized while preserving the integrity of critical data. Meanwhile, efficient communication methods, such as peer-to-peer exchanges, are often implemented to reduce data transfer times.

However, managing large-scale graphs becomes considerably more challenging when dealing with dynamic graphs. For example, if we employ mini-batch training techniques, the model must maintain a sufficient history horizon, optimize the memory update mechanism for nodes, and transfer information using memory synchronization after partitioning into subgraphs. For temporal sequential processing, the main approaches include using time encoding to capture temporal information, generating batches of data in chronological order, and ensuring that all data are traversed in an overall sequential manner. Collectively, these methodologies and fundamental prerequisites contribute to enhancing parallelism, scalability, and efficient processing capabilities for large dynamic graphs.

Existing approaches can be grouped into two primary lines. One approach gives precedence to scalability in the design of dynamic GNNs, resulting in various algorithms aimed at efficient learning on dynamic graphs. The second focuses on the development of scalable training frameworks that enable the adaptation of existing algorithms to large-scale dynamic graphs.

4.1 Scalable GNNs for Dynamic Graphs

Instead of adapting training frameworks to fit existing algorithms for large-scale dynamic graph learning, several studies have suggested developing algorithms specifically designed for large-scale dynamic graphs. To ensure the efficiency and scalability of the model, techniques like sampling and

Table 3 Scalable Dynamic GNNs training framework.

Name	Graph Type	Hardware	Software
PiPAD [69]	DTDG	single-node single-GPU with multi-core CPU	PyG Temporal
DGNN-Booster [70]	DTDG	single-node single-GPU with multi-core CPU	Pytorch
Chakaravarthy et al. [71]	DTDG	multi-node multi-GPU	PyTorch
TGL [72]	CTDG	single-node multi-GPU	DGL
DistTGL [73]	CTDG	multi-node multi-GPU	DGL
SPEED [74]	CTDG	single-node multi-GPU	Pytorch
Xia et al. [75]	CTDG	single-node multi-GPU	PyTorch
STEP [76]	CTDG	single-node single-GPU with multi-core CPU	PyTorch

parallel training are often integrated for optimization. Following this framework, SEIGN [45] employs a parameter-free messaging strategy for preprocessing and uses the Inception architecture to obtain multi-scale node representations. Similarly to EvolveGCN [44], SEIGN updates the convolutional kernel weights with GRU modules for each graph snapshot to maintain their freshness. However, instead of evolving the entire GNN model, SEIGN refines only the parameters of the convolutional kernel, thereby enhancing the method's scalability. Furthermore, SEIGN introduces the evolution of node representations, increasing the expressiveness of the final node representation. Additionally, it facilitates efficient graph mini-batch training, further enhancing the model's scalability.

EARLY [58] reduces computational costs by updating only the top-k nodes most susceptible to event impacts to meet the frequent changes in CTDGs. The selection of these top-k nodes is guided by two primary metrics: local and global scores. Local scores emphasize the theoretical analysis of graph convolution networks, addressing both the effect of node updates and the variations in random walk probabilities. On the other hand, global scores emphasize the node influence. Ultimately, an information gain function integrates these scores to identify the most influential top-k nodes.

Inspired by scalable GNNs on static graphs [77], Zheng et al. [59] proposed a decoupled GNN model that separates the graph propagation process and the training process for the downstream tasks, where the unified dynamic propagation process can efficiently handle both continuous and discrete dynamic graphs. By leveraging the identity equation of the graph propagation process, the model locates affected nodes and quantifies the impact on their representations from graph changes. Other node representations are adjusted in the next step. This approach allows incremental updates from prior results, avoiding costly recalculations. Additionally, since computations related to the graph structure only occur during propagation, subsequent training for downstream tasks can be performed separately without involving expensive graph operations. Therefore, arbitrary sequence learning models can be plugged in. Similarly, SDG [60] employs a dynamic propagation matrix based on personalized PageRank, replacing the static probability transition matrix of APPNP [78]. Specifically, SDG first extracts hidden node features using a model-agnostic neural network. These features are then multiplied with the dynamic propagation matrix to disseminate neighborhood node information. The dynamic propagation matrix tracks the steady-state distribution of random walks following graph topology

changes using the push-out and add-back algorithm, eliminating redundant propagation computations in dynamic graphs.

4.2 Scalable Training Frameworks for Dynamic GNNs

Frameworks designed for DTDGs. Memory management optimization to support efficient model parallelism is a prevalent method for enhancing the training efficacy of GNNs for DTDGs. To optimize the end-to-end training performance of dynamic graph neural networks on a single GPU, Pi-PAD [69] introduces a pipeline execution framework and multi-snapshot parallel processing manner. Additionally, it employs a slice-based graph representation to efficiently extract overlapping sections of the graph, enabling parallel computations for multiple snapshots simultaneously. DGNN-Booster [70] uses the Field-Programmable Gate Array (FPGA) to enhance the inference speed of dynamic graph neural networks in a single-machine setting. It also introduces two distinct dataflow designs to support mainstream DGNN models. Chakaravarthy et al. [71] employed optimization techniques such as gradient checkpointing and graph-difference-based CPU-GPU data transfer to enhance the training of dynamic GNNs on single-node multi-GPU systems. Additionally, they introduced an algorithm that uses snapshot partitioning to train large-scale dynamic GNNs on distributed multi-node multi-GPU systems efficiently. Experimental results indicate that this method provides notable improvements over traditional methods.

Frameworks designed for CTDGs. To overcome the limitations of single-machine configurations and further improve the scalability of dynamic graph neural network models for processing large dynamic graphs, more prevalent strategies involve partitioning the graph across multiple devices for parallel training. In this context, random division and vertex-based partitioning are frequently employed. TGL [72] proposes a generalized framework to train different dynamic GNNs, including snapshot-based, time encoding-based and memory-based methods. Temporal-CSR data structures are designed to enable fast access to temporal edges, while parallel samplers support efficient implementation of various temporal neighbor sampling algorithms. To overcome the timeliness issue of node memory when training with large batches, a random chunk scheduling technique is proposed, which enables efficient multi-GPU training on large-scale dynamic graphs. Numerous dynamic GNNs maintain node-level memory vectors to summarize historical information about each node. To maintain and update these node memories efficiently, TGL designs the mailbox to cache the latest messages. In each training iteration, the latest messages are read from the mailbox, aggregated by a certain combiner. Then, the memory state of the node is updated by a sequential model such as GRU. This design circumvents the direct generation of mail using the most recent memory state, effectively avoiding potential information leakage. Additionally, it incorporates an asynchronous mechanism for updating node states, making it better aligned with the streaming or batch training process.

To further enhance efficiency, DistTGL [73] implements a mechanism where memory operations are serialized and executed asynchronously through a separate daemon process, circumventing intricate synchronization requirements. DistTGL [73] incorporates an additional static node memory within the dynamic GNNs, which not only enhances the model's accuracy but also accelerates its convergence rate. Two novel training methodologies, namely

time period parallelism and memory parallelism, are also proposed. These two methodologies enable capturing a comparable number of graph event dependencies on multiple GPUs, similar to that achieved in single GPU training. Prefetching and pipelining techniques are utilized to mitigate the associated costs of batch formation and to facilitate overlap with GPU training. The DistTGL [73] framework offers notable enhancements in terms of convergence speed and training throughput compared to TGL [72]. This enables the efficient scaling of memory-based dynamic GNNs training on distributed GPU clusters, achieving near-linear speedup ratios. The guidance on the identification of the most effective training configurations is provided by employing a variety of parallelization algorithms that are tailored to the specific dataset and hardware attributes.

In contrast, the SPEED [74] framework introduces a streaming edge partitioning module that integrates temporal information through an exponential decay technique. Additionally, it effectively minimizes the replication ratio by regulating the number of shared nodes. A vertex partitioning method based on interval properties is also employed, combined with joint and asynchronous pipelining techniques between mini-batches. This ensures efficient parallel training on multiple GPUs for large-scale dynamic graphs. The design speeds up the training and reduces memory consumption on a single GPU significantly.

Liu et al. [75] proposed a method that emphasizes hierarchical pipeline parallelism, integrating data prefetching, joint pipelining across mini-batches, and asynchronous pipelining. Additionally, to enhance communication efficiency, they introduced a strategy for partitioning graphs and vertices to eliminate redundancy. Another framework, STEP [76],

provides an unsupervised pruning method for large-scale dynamic graphs. Node and edge representations are acquired by self-supervised adversarial learning subsequent to each graph update. Moreover, it selects edges for sampling depending on their significance, resulting in a graph known as the undermine graph. STEP utilizes the graph provided to train a basic pruning network. This enables the STEP framework to effectively manage the process of removing unnecessary elements from dynamic graphs, during both the training and inference stages, with a particular focus on newly added edges.

In general, these advanced methods jointly enhance the effective parallel training of dynamic graph neural networks in distributed settings, each with its distinct benefits and potential use cases, as shown in Table 3.

5 Datasets and Benchmarks Overview

5.1 Datasets

In recent years, the quantity and variety of dynamic graph datasets made available to the public have increased significantly. These datasets provide valuable resources for researchers to validate and compare algorithms, thereby facilitating the exploration of new research directions. Table 4 contains basic characteristics of commonly used dynamic graph datasets. We can divide these datasets into CTDGs and DTDGs, according to the collection and storage methods of the datasets. However, with appropriate processing, they can be transformed into one another. From these datasets, we can observe the following characteristics:

- **Domains and applications.** These datasets cover a diverse range of domains, including social, music, political, transportation, economic, and e-

Table 4 The statistics of datasets, where n represents the number of nodes, m denotes the number of edges, $|T|$ indicates the number of timestamps or snapshots, and d_e refers to the dimension of edge features.

Types	Dataset	Domain	Category	n	m	$|T|$	d_e
CTDG	Social Evolution [79]	Proximity	general	74	2,099,519	565,932	1[†]
	Enron [80]	Social	general	184	125,235	22,632	-
	Contact [81]	Proximity	general	692	2,426,279	8,065	1[†]
	UCI [82]	Social	general	1,899	59,835	58,911	100
	LastFM [54]	Interaction	bipartite	1,980	1,293,103	1,283,614	-
	Bitcoin-Alpha [83, 84]	Finance	general	3,783	24,186	24,186	-
	Bitcoin-OTC [83, 84]	Finance	general	5,881	35,592	35,592	-
	MOOC [54]	Interaction	bipartite	7,144	411,749	345,600	4[‡]
	Wikipedia [54]	Social	bipartite	9,227	157,474	152,757	172
	Reddit [54]	Social	bipartite	10,984	672,447	669,065	172
	GDELT [72, 85]	Event	general	16,682	191,290,882	170,522	186
	eBay-Small [86]	E-commerce	bipartite	38,427	384,677	-	-
	Taobao [87, 88]	E-commerce	bipartite	82,566	77,436	-	4[‡]
	YouTube-Reddit-Small [89]	Social	bipartite	264,443	297,732	-	-
	eBay-Large [86]	E-commerce	bipartite	1,333,594	1,119,454	-	-
	Taobao-Large [87, 88]	E-commerce	bipartite	1,630,453	5,008,745	-	4[‡]
	DGraphFin [90]	Social	general	3,700,550	4,300,999	-	-
	Youtube-Reddit-Large [89]	Social	bipartite	5,724,111	4,228,523	-	-
DTDG	UN Vote [91]	Politics	general	201	1,035,742	72	1[†]
	US Legislative [92, 93]	Politics	general	225	60,396	12	1[†]
	UN Trade [94]	Finance	general	255	507,497	32	1[†]
	Canadian Parliament [92]	Politics	general	734	74,478	14	-
	Twitter-Tennis [95]	Social	general	1,000	40,839	120	-
	Autonomous systems [96]	Communication	general	7,716	13,895	733	-
	Flights [97]	Transport	general	13,169	1,927,145	122	1[†]
	HEP-TH [96, 98]	Citation	general	27,770	352,807	3,487	-
	Elliptic [99]	Finance	general	203,769	234,355	49	-
	MAG [11, 72]	Citation	general	121,751,665	1,297,748,926	120	-

[†] Represents the edge weight.

[‡] Indicates the number of different behaviors.

commerce networks. This diversity enables researchers to test and validate algorithms on various real-world applications.

- **Scale.** The datasets vary in graph sizes, ranging from small-scale graphs with thousands of nodes and edges (e.g., UCI [82] and Contact [81]) to large-scale graphs with millions or even billions of nodes (e.g. Taobao [87, 88] and MAG [72]). This indicates that existing datasets can support research and applications at different scales.

- **Types.** Many datasets are bipartite graphs. For example, datasets such as MOOC [54], LastFM [54], eBay [86], and TaoBao [86] fall into this category, with nodes categorized into two distinct sets and edges representing interactions between these sets. In contrast, some datasets, like Enron [80] and SocialEvo [100], are general graphs.

- **Features.** Although these datasets provide rich information about nodes and edges, many of them do not provide features of nodes or edges. Only

a few datasets like Reddit [54], Wikipedia [54] and UCI [82] provide edge features, which can help researchers gain deeper insights into inter-action patterns between nodes. Datasets offering node features are even rarer, with GDELT [72], Elliptic [99], and MAG [72] being notable exceptions. Conversely, datasets such as LastFM [54], Enron [80], and Flights [97] lack node feature details, potentially due to privacy concerns or the sensitive nature of the raw data. Moreover, in practical applications, identifying or defining appropriate features can also be challenging. Overall, the currently available public datasets provide diverse resources for representation learning on dynamic graphs. However, these datasets also exhibit some common characteristics, such as the lack of node and edge features, which pose challenges and opportunities for researchers. This is because collecting features in real-world applications is often costly, and learning patterns solely from graph structure changes can improve model generalization. It would be very encouraging if new high-quality public datasets with extensive features were to emerge in the future, as this would facilitate more research projects.

5.2 Prediction Problems

Graph learning tasks include three levels: node-level, edge-level, and graph-level [11]. In node-level tasks, the primary focus is on the attributes of individual nodes, which include node classification, regression, clustering, and so on. Edge-level tasks emphasize information about the edge. For example, link prediction determines whether an edge exists between two nodes, while link classification identifies the type or attributes of an edge. Graph-level tasks analyze the entire structure, such as classifying the category of the graph (graph classification) or predicting its continuous values (graph regression). Graph-generation tasks aim to generate new graphs. In addition, there are specialized tasks, such as graph generation, whose objective is to generate entirely new graph structures based on specific criteria or patterns observed in existing data.

The three levels of tasks also exist in dynamic graphs. The primary difference from the static context is that the graph changes over time, resulting in simultaneous alterations to the properties of nodes, edges, and the entire graph. For node-level tasks, attributes and connection patterns of nodes may change, potentially impacting the results of classification or regression analyses. In edge-level tasks, the presence and attributes of edges may shift over time, for example, new connections can emerge while old ones disappear. For graph-level tasks, the topology of the graph might become more intricate with the addition of new nodes and edges, which then impact results in graph classification, regression, and generation. Next, we will explore specific application scenarios of dynamic graph analytics using link prediction, anomaly detection, and community discovery as examples.

Link prediction in dynamic graphs aims to predict potential edges in future graphs based on connections from previous time steps, also known as temporal link prediction. This requires not only the structural features of the current graph but also identifying trends from its dynamic changes. Link prediction has a wide range of applications. In citation networks, by examining both authors' evolving citation behaviors and changes in research focus, we can predict the authors they might reference in future publications.

Anomaly detection in dynamic graphs can be categorized into edge and node anomaly detection. Edge anomaly detection aims to identify edges in a

dynamic graph that deviate from historical evolution patterns. Such irregularities may arise from connections that rogue users have introduced or manipulated maliciously. Conversely, node anomaly detection focuses on identifying nodes that distinctly differ from their peers or exhibit atypical behaviors. Anomaly detection has broad applications, particularly in fraud detection and e-commerce. For example, on e-commerce sites, users might manipulate the system by repeatedly clicking on specific products to increase their popularity artificially. They could also simultaneously click on a product and another trendy item, deceiving the recommendation system into overestimating the similarity between the two items. These connections are considered anomalous edges, and the deceptive users are considered anomalous nodes.

Community detection is a fundamental task in understanding complex graphs. Community structures are commonly observed in real-world networks that encompass various domains, such as social, biological, or technological. Based on the density of connections among nodes, the graph can be divided into various communities. Typically, nodes within a community have denser connections, while connections between nodes from different communities are less frequent. This community structure is also present in dynamic graphs, but the community configurations can evolve as the graph structure changes. The identification and comprehension of dynamic community structures and their evolutionary patterns can provide a deeper comprehension of the graph and its community formations, which facilitates the prediction of potential changes in the graph. Taking social networks as an example, users with similar likes and follows can be grouped into the same interest clusters. By identifying these clusters, the system is able to recommend new connections or content that aligns with their interests.

Transductive / inductive setting. In the transductive setting, the model attempts to make accurate predictions on the graph on which it was initially trained. It trains on a specific portion of the graph while being fully aware of the entire structure, including the unlabeled portions. In contrast, in the inductive setting, the model learns from a designated training graph to make reliable predictions on independent, completely unknown graphs. In dynamic graphs, where nodes and edges frequently change over time, the essence of the inductive setting is the model's ability to effectively generalize to unseen nodes and dynamically evolving structures.

Interpolation / extrapolation. When predicting the value of a function in an unknown region based on existing data and models, predictions within the range of existing data are called interpolations, while those outside are termed extrapolations. Taking dynamic node classification as an example, extrapolation involves predicting future states ($> t$) based on observations up to time t, such as in weather forecasting. The interpolation setting involves predicting states within the observed range (t_0, t) based on observations up to time t, like completing missing values. Similarly, in link prediction, extrapolation predicts whether future edges (v_i, v_j) will exist, while interpolation determines if edges (v_i, v_j) exist within the observed range (t_0, t), as in knowledge graph completion. In summary, interpolation involves predicting unobserved values within existing data, whereas extrapolation predicts beyond the observed range. For dynamic graphs, extrapolation is used for future predictions, while interpolation focuses on imputing missing information within observations, depending on whether the target concerns observed or unobserved future states.

Table 5   Links of benchmarks

Benchmark	Code
PyTorch Geometric Temporal	https://github.com/benedekrozemberczki/pytorch_geometric_temporal
TGL	https://github.com/amazon-science/tgl
Dynamic Graph Library (DyGLib)	https://github.com/yule-BUAA/DyGLib
Temporal Graph Benchmark (TGB)	https://github.com/shenyangHuang/TGB
BenchTemp	https://github.com/qianghuangwhu/benchtemp

5.3   Benchmarks

In the past, different methods had their own configurations and implementations, complicating the direct comparison of model performances. Recent work has tried to establish unified frameworks for training and evaluation to rectify this. By providing standardized datasets, synchronized preprocessing stages, and fair training strategies, these frameworks make sure that all methods are tested in the same way. Researchers can then easily compare different models for the same tasks, pinpointing the merits and shortcomings of each approach with greater clarity. These efforts demonstrate a prevailing tendency towards heightened standardization in the field of dynamic graphs, which benefits the evaluation and comparison of different methods. Table 5 provides links to these benchmarks.

6   Advanced Techniques in Graph Processing

6.1   Dynamic Transfer Learning

Transfer learning employs the knowledge of a model trained on one task (called the source task) to facilitate the learning of another distinct but related task (called the target task). The core idea is to share specific knowledge between the source and the target task. This concept is based on the observation that various tasks frequently share underlying features or patterns. Consequently, insights from one task can potentially benefit another [101, 102]. Dynamic transfer learning is a transfer learning setting proposed for scenarios where both the source and target domains evolve. In this context, the current state of the target domain depends on its historical state, which cannot be transferred straightforwardly. Thus, dynamic transfer learning requires continuous real-time adjustments to the shifting data distributions in both the source and target domains. It also demands the prompt transfer of pertinent knowledge from the evolving source domain to the concurrently changing target domain, aiming to optimize performance in the target tasks. The primary challenge lies in effectively modeling inter-domain evolution and managing temporal interdependencies across various time steps.

Dynamic transfer learning was first introduced to address diverse real-world challenges in the areas of computer vision and natural language processing. Recently, researchers have tried to explore its potential in graph learning. As of now, research into dynamic transfer learning for graphs is still in its early stages. It mainly involves applying ideas from fields other than graphs, like using LSTM or attention mechanisms to model the graph's temporal pattern [103]. In domains outside of graph theory, substantial research has been conducted on dynamic source domain non-stationarity, target domain concept drift, and related topics. Future investigations should delve deeper into the specific attributes of graph data, aiming to design more suitable dynamic transfer learning methods. This

will further position dynamic transfer learning as a pivotal tool in addressing various time-sensitive practical challenges.

6.2 Pretraining Techniques for Dynamic GNNs

Typically, pre-trained models are trained on large-scale datasets to obtain universal feature representations, which are then fine-tuned on specific downstream tasks to enhance their performance. Originating from the fields of computer vision and natural language processing, pre-trained models have achieved significant successes in many tasks, such as semantic parsing, image classification and image segmentation. With the increase in the volume of graph data, pre-training models have recently been introduced to the domain of graph learning. Related work usually employs self-supervised strategies to design pre-training tasks such as edge generation [104], attribute masking [104] and subgraph comparison [105]. Since nodes and edges in dynamic graphs vary over time, these models face the issue of describing the temporal dependence. Therefore, PT-DGNN [106] uses a time-based masking strategy to predict the most recent edges or attributes, learning the temporal evolution law of the graph. On the other hand, CPDG [107] contrasts pairs of positive and negative samples. CPDG captures patterns both temporally and structurally by enhancing the similarity between positive samples and diminishing it among negative samples. Additionally, CPDG stores memory states from different time steps to help the model preserve long-term evolutionary information, further supporting subsequent downstream tasks.

However, there are still numerous challenges that require additional research. Scalability is a major issue in this field, and the key question is how to train models in advance so that they can effi-ciently deal with dynamic graphs at different temporal granularities and still be flexible enough to perform many different downstream tasks. In addition, dynamic graphs contain both long-term stability patterns and short-term variability patterns, which means the model needs to adjust for both. Furthermore, the structural neighbor information of each node is essential, so the model is required to preserve this structural information effectively. In real-world applications, pre-training for large-scale dynamic graphs requires efficient sampling and training mechanisms. Simultaneously, the temporal patterns learned by the pre-trained models should be promptly incorporated into subsequent tasks.

In summary, current research and applications primarily focus on designing appropriate pre-training tasks, sampling strategies, and contrastive learning methods for dynamic graphs. The objective is to achieve resilient temporal and structural depictions that improve the adaptability of the model for various tasks. Simultaneously, it is important to include enduring and immediate trends effectively, and then improve the efficacy of model training and transfer.

7 Future Trends and Challenges

7.1 More Complex Dynamic Graphs

Although some large-scale dynamic graph datasets have emerged in recent years, dynamic graph datasets are still lacking in diversity compared to static graph datasets. This may lead to insufficient model performance in diverse, real-world, dynamic graph applications.

- Firstly, the results of EdgeBank [100] indicate that in some cases, using only historical information can achieve good results. This suggests that the complexity of current datasets may be

insufficient to distinguish between different models significantly. Therefore, we need to construct more challenging datasets containing richer structural information and temporal evolution patterns to evaluate models' modeling capabilities better.

- Secondly, operations like users unfollowing others are shared in real applications, but currently, there is a lack of real-world datasets containing such edge deletion data. Expanding datasets could involve collecting and integrating data with real edge deletion operations to simulate the dynamics of graph data in the real world. This can help assess models' robustness and performance when handling edge deletion.

- Finally, some special graphs are rarely mentioned in current works, such as dynamic signed networks [108] and dynamic heterophilic graphs. In the future, dynamic graph datasets containing different network topologies (e.g. small-world, scale-free) could be constructed, and more special graphs could be collected and generated.

In summary, larger-scale, richer-content, and more complex structure datasets are needed to facilitate the development of dynamic graph learning towards handling real-world complex networks.

7.2 Interpretability of Dynamic GNNs

The frameworks and methods related to deep model interpretability are designed to help users understand and trust these models, enabling them to be applied more effectively. There have been some advances in the area of interpretability for static GNN models [109], while research on interpretability for dynamic GNNs appears to still be in its early stages. Xie et al. [110] proposed DGExplainer, which aims to provide a reliable interpretation of dynamic GNNs. DGExplainer redistributes the output activation scores of a dynamic GNN to the neurons of the previous layer to calculate the relevance scores of the input neurons, thereby identifying important nodes in link prediction and node regression tasks. The DynBrainGNN [111] model, designed to provide both prediction and explanation, contains built-in interpretable GNNs with dynamic FCs for psychiatric disease analysis. These studies show that although the interpretability of dynamic GNNs is a relatively new and less discussed area, there are already some studies attempting to address this issue.

7.3 LLMs for dynamic graph learning

Recently, large language models (LLMs) have shown remarkable capabilities in natural language processing, achieving breakthroughs in many aspects, such as text generation [112], question answering [113], and dialogue tasks [114]. Researchers have tried to adapt LLM to more fields to utilize its powerful pre-trained language understanding ability. LLM has strong few-shot, in-context learning ability, and by designing prompting techniques carefully, it can help dynamic graph learning models to understand spatial and temporal information better. For example, LLM4DyG [115] examines that providing only one example can significantly improve the accuracy of LLM on multiple dynamic graph tasks such as "when link", "when connect", "when triadic closure", "neighbor at time" and "neighbor in periods". On the other hand, LLM can perform reasoning in a given context and solve dynamic graph related tasks. This insight motivates new applications of LLMs in dynamic graph learning tasks and potentially transforms approaches in related fields. Although LLMs have demonstrated inspiring understanding capabilities on dynamic graph tasks, they still face challenges as follows.

- **Processing large-scale graphs**: As graph size

grows, the performance of LLMs markedly declines [116]. The reason may be that the increase in graph size makes the amount of contextual information soaring, which makes it difficult for LLM to capture the key information. The ability of LLMs to model large-scale dynamic graphs remains to be validated.

- **Computational efficiency**: Current LLMs are mostly large and computationally expensive. How to improve the computational speed of LLMs and improve their efficiency in learning dynamic graphs is an issue that needs to be addressed.

- **Lack of structural modeling**: LLMs lack the capability to explicitly model graph structures in the same way that GNN does. Merely relying on contextual descriptions to acquire information may not be adequate and direct. Therefore, additional investigation is required to improve LLMs' ability to model graph topology.

8 Conclusion

Dynamic graph neural networks, a recent development in representation learning, can process changing graph structures and have potential applications across various fields. In this paper, we conduct a comprehensive survey on the key issues of dynamic GNNs, including prediction tasks, temporal modeling methods, and the handling of large-scale dynamic graphs. We have summarized the main advancements in current research and pointed out some promising key directions, such as expanding the scope of existing models and researching the expressive capability of dynamic GNNs. Dynamic GNNs still face challenges, such as scalability, theoretical guidance, and dataset construction. We hope that this paper can provide valuable references for researchers in this field, and promote the further development of theories and methods related to

dynamic GNNs. An important direction for future research is to continue expanding the applicability of dynamic graph representation learning, handling more complex and diverse dynamic graph structures and evolution patterns, and making it a unified representation learning framework with both breadth and depth. As more and more practical applications involve dynamic graph analysis, dynamic GNNs will undoubtedly become a continuously active and increasingly important research area.

References

1. Kazemi S M, Goel R, Jain K, Kobyzev I, Sethi A, Forsyth P, Poupart P.   Representation learning for dynamic graphs: A survey. The Journal of Machine Learning Research, 2020, 21(1): 2648–2720

2. Xu d , ruan c , korpeoglu e , kumar s , achan k . Inductive representation learning on temporal graphs. In: International Conference on Learning Representations. 2020

3. Rossi E, Chamberlain B, Frasca F, Eynard D, Monti F, Bronstein M.   Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637, 2020

4. You J, Du T, Leskovec J. Roland: graph learning framework for dynamic graphs. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2022, 2358–2366

5. Zhu Y, Lyu F, Hu C, Chen X, Liu X. Encoder-decoder architecture for supervised dynamic graph learning: A survey. arXiv preprint arXiv:2203.10480, 2022

6. Barros C D, Mendonca M R, Vieira A B, Ziviani A. A survey on embedding dynamic graphs. ACM Computing Surveys, 2021, 55(1): 1–37

7. Cai B, Xiang Y, Gao L, Zhang H, Li Y, Li J. Temporal knowledge graph completion: A survey. arXiv preprint arXiv:2201.08236, 2022

8. Liu C, Paterlini S. Stock price prediction using temporal graph model with value chain data. arXiv preprint arXiv:2303.09406, 2023

9. Wang X, Ma Y, Wang Y, Jin W, Wang X, Tang J, Jia C, Yu J. Traffic flow prediction via spatial temporal graph neural network. In: Proceedings of the Web Conference. 2020, 1082–1092

10. Gao Y, Wang X, He X, Feng H, Zhang Y. Rumor detection with self-supervised learning on texts and social graph. Frontiers of Computer Science, 2023, 17(4): 174611

11. Hu W, Fey M, Ren H, Nakata M, Dong Y, Leskovec J. Ogb-lsc: A large-scale challenge for machine learning on graphs. arXiv preprint arXiv:2103.09430, 2021

12. Fu D, He J. Dppin: A biological repository of dynamic protein-protein interaction network data. In: IEEE International Conference on Big Data. 2022, 5269–5277

13. Hawkes A G. Spectra of some self-exciting and mutually exciting point processes. Biometrika, 1971, 58(1): 83–90

14. Zuo S, Jiang H, Li Z, Zhao T, Zha H. Transformer hawkes process. In: International Conference on Machine Learning. 2020, 11692–11702

15. Lu Y, Wang X, Shi C, Yu P S, Ye Y. Temporal network embedding with micro-and macro-dynamics. In: Proceedings of the ACM International Conference on Information and Knowledge Management. 2019, 469–478

16. Zuo Y, Liu G, Lin H, Guo J, Hu X, Wu J. Embedding temporal network via neighborhood formation. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2018, 2857–2866

17. Lu Y, Wang X, Shi C, Yu P S, Ye Y. Temporal network embedding with micro-and macro-dynamics. In: Proceedings of the ACM International Conference on Information and Knowledge Management. 2019, 469–478

18. Roweis S T, Saul L K. Nonlinear dimensionality reduction by locally linear embedding. Science, 2000, 290(5500): 2323–2326

19. Belkin M, Niyogi P. Laplacian eigenmaps for dimensionality reduction and data representation. Neural Computation, 2003, 15(6): 1373–1396

20. Cao S, Lu W, Xu Q. Grarep: Learning graph representations with global structural information. In: Proceedings of the ACM International Conference on Information and Knowledge Management. 2015, 891–900

21. Zhu D, Cui P, Zhang Z, Pei J, Zhu W. High-order proximity preserved embedding for dynamic networks. IEEE Transactions on Knowledge and Data Engineering, 2018, 30(11): 2134–2144

22. Bartunov S, Kondrashkin D, Osokin A, Vetrov D. Breaking sticks and ambiguities with adaptive skip-gram. In: Artificial Intelligence and Statistics. 2016, 130–138

23. Joachims T. Text categorization with support vector machines: Learning with many relevant features. In: European Conference on Machine Learning. 1998, 137–142

24. Perozzi B, Al-Rfou R, Skiena S. Deepwalk: Online learning of social representations. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2014, 701–710

25. Grover A, Leskovec J. Node2vec: Scalable feature learning for networks. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016, 855–864

26. Wang D, Cui P, Zhu W. Structural deep network embedding. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016, 1225–1234

27. Cao S, Lu W, Xu Q. Deep neural networks for learning graph representations. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2016

28. Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. Advances in Neural Information Processing Systems, 2016, 29

29. Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations. 2016

30. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. In: International Conference on Learning Representations. 2018

31. Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. Advances in Neural Information Processing Systems, 2017, 30

32. Zhu D, Cui P, Zhang Z, Pei J, Zhu W. High-

order proximity preserved embedding for dynamic networks. IEEE Transactions on Knowledge and Data Engineering, 2018, 30(11): 2134–2144

33. Li J, Dani H, Hu X, Tang J, Chang Y, Liu H. Attributed network embedding for learning in a dynamic environment. In: Proceedings of the ACM International Conference on Information and Knowledge Management. 2017, 387–396

34. Nguyen G H, Lee J B, Rossi R A, Ahmed N K, Koh E, Kim S. Continuous-time dynamic network embeddings. In: Proceedings of the Web Conference. 2018, 969–976

35. Goyal P, Chhetri S R, Canedo A. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowledge-Based Systems, 2020, 187: 104816

36. Manessi F, Rozza A, Manzo M. Dynamic graph convolutional networks. Pattern Recognition, 2020, 97: 107000

37. Sankar A, Wu Y, Gou L, Zhang W, Yang H. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In: Proceedings of the International Conference on Web Search and Data Mining. 2020, 519–527

38. Wang Y, Li P, Bai C, Subrahmanian V, Leskovec J. Generic representation learning for dynamic social interaction. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Workshop. 2020

39. Wang Y, Li P, Bai C, Leskovec J. Tedic: Neural modeling of behavioral patterns in dynamic social interaction networks. In: Proceedings of the Web Conference. 2021, 693–705

40. Chen J, Wang X, Xu X. Gc-lstm: Graph convolution embedded lstm for dynamic network link prediction. Applied Intelligence, 2022, 1–16

41. Li J, Han Z, Cheng H, Su J, Wang P, Zhang J, Pan L. Predicting path failure in time-evolving graphs. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2019, 1279–1289

42. Jin W, Qu M, Jin X, Ren X. Recurrent event network: Autoregressive structure inferenceover temporal knowledge graphs. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. 2020, 6669–6683

43. Zhu Y, Cong F, Zhang D, Gong W, Lin Q, Feng W, Dong Y, Tang J. Wingnn: Dynamic graph neural networks with random gradient aggregation window. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2023, 3650–3662

44. Pareja A, Domeniconi G, Chen J, Ma T, Suzumura T, Kanezashi H, Kaler T, Schardl T, Leiserson C. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 5363–5370

45. Qin X, Sheikh N, Lei C, Reinwald B, Domeniconi G. Seign: A simple and efficient graph neural network for large dynamic graphs. In: IEEE International Conference on Data Engineering. 2023, 2850–2863

46. Goyal P, Chhetri S R, Canedo A. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowledge-Based Systems, 2020, 187: 104816

47. Trivedi R, Dai H, Wang Y, Song L. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In: International Conference on Machine Learning. 2017, 3462–3471

48. Trivedi R, Farajtabar M, Biswal P, Zha H. Dyrep: Learning representations over dynamic graphs. In: International Conference on Learning Representations. 2019

49. Knyazev B, Augusta C, Taylor G W. Learning temporal attention in dynamic graphs with bilinear interactions. Plos One, 2021, 16(3): e0247936

50. Han Z, Ma Y, Wang Y, Günnemann S, Tresp V. Graph hawkes neural network for forecasting on temporal knowledge graphs. In: Automated Knowledge Base Construction. 2020

51. Sun H, Geng S, Zhong J, Hu H, He K. Graph hawkes transformer for extrapolated reasoning on temporal knowledge graphs. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. 2022, 7481–7493

52. Wen Z, Fang Y. Trend: Temporal event and node dynamics for graph representation learning. In: Proceedings of the ACM Web Conference. 2022, 1159–1169

53. Ma Y, Guo Z, Ren Z, Tang J, Yin D. Streaming graph neural networks. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020, 719–728

54. Kumar S, Zhang X, Leskovec J. Predicting dynamic embedding trajectory in temporal interaction networks. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2019, 1269–1278

55. Wang X, Lyu D, Li M, Xia Y, Yang Q, Wang X, Wang X, Cui P, Yang Y, Sun B, others . Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In: Proceedings of the International Conference on Management of Data. 2021, 2628–2638

56. Wang Y, Chang Y Y, Liu Y, Leskovec J, Li P. Inductive representation learning in temporal networks via causal anonymous walks. In: International Conference on Learning Representations. 2020

57. Li Y, Shen Y, Chen L, Yuan M. Zebra: When temporal graph neural networks meet temporal personalized pagerank. Proceedings of the VLDB Endowment, 2023, 16(6): 1332–1345

58. Li H, Chen L. Early: Efficient and reliable graph neural network for dynamic graphs. Proceedings of the ACM on Management of Data, 2023, 1(2): 1–28

59. Zheng Y, Wei Z, Liu J. Decoupled graph neural networks for large dynamic graphs. Proceedings of the VLDB Endowment, 2023, 16(9): 2239–2247

60. Fu D, He J. Sdg: A simplified and dynamic graph neural network. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval. 2021, 2273–2277

61. Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. Advances in Neural Information Processing Systems, 2016, 29

62. Bonner S, Atapour-Abarghouei A, Jackson P T, Brennan J, Kureshi I, Theodoropoulos G, McGough A S, Obara B. Temporal neighbourhood aggregation: Predicting future links in temporal graphs via recurrent variational graph convolutions. In: IEEE International Conference on Big Data. 2019, 5336–5345

63. Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014

64. Goyal P, Kamra N, He X, Liu Y. Dyngem: Deep embedding method for dynamic graphs. arXiv preprint arXiv:1805.11273, 2018

65. Chen T, Goodfellow I, Shlens J. Net2net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641, 2015

66. Kipf T, Fetaya E, Wang K C, Welling M, Zemel R. Neural relational inference for interacting systems. In: International Conference on Machine Learning. 2018, 2688–2697

67. Kazemi S M, Goel R, Eghbali S, Ramanan J, Sahota J, Thakur S, Wu S, Smyth C, Poupart P, Brubaker M. Time2vec: Learning a vector representation of time. arXiv preprint arXiv:1907.05321, 2019

68. Loomis L H. Introduction to abstract harmonic analysis. Courier Corporation, 2013

69. Wang C, Sun D, Bai Y. Pipad: Pipelined and parallel dynamic gnn training on gpus. In: Proceedings of the ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming. 2023, 405–418

70. Chen H, Hao C. Dgnn-booster: A generic fpga accelerator framework for dynamic graph neural network inference. arXiv preprint arXiv:2304.06831, 2023

71. Chakaravarthy V T, Pandian S S, Raje S, Sabharwal Y, Suzumura T, Ubaru S. Efficient scaling of dynamic graph neural networks. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021, 1–15

72. Zhou H, Zheng D, Nisa I, Ioannidis V, Song X, Karypis G. Tgl: a general framework for temporal gnn training on billion-scale graphs. Proceedings of the VLDB Endowment, 2022, 15(8): 1572–1580

73. Zhou H, Zheng D, Song X, Karypis G, Prasanna V. Disttgl: Distributed memory-based temporal graph neural network training. arXiv preprint arXiv:2307.07649, 2023

74. Chen X, Liao Y, Xiong Y, Zhang Y, Zhang S, Zhang J, Sun Y. Speed: Streaming partition and parallel acceleration for temporal interaction graph embedding. arXiv preprint arXiv:2308.14129, 2023

75. Xia Y, Zhang Z, Wang H, Yang D, Zhou X, Cheng D. Redundancy-free high-performance dynamic gnn training with hierarchical pipeline parallelism. In: Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing. 2023, 17–30

76. Li J, Tian S, Wu R, Zhu L, Zhao W, Meng C, Chen L, Zheng Z, Yin H. Less can be more: Unsupervised graph pruning for large-scale dynamic graphs. arXiv preprint arXiv:2305.10673, 2023

77. Wu F, Souza A, Zhang T, Fifty C, Yu T, Weinberger K. Simplifying graph convolutional networks. In: International Conference on Machine Learning. 2019, 6861–6871

78. Gasteiger J, Bojchevski A, Günnemann S. Predict then propagate: Graph neural networks meet personalized pagerank. In: International Conference on Learning Representations. 2018

79. Madan A, Cebrian M, Moturu S, Farrahi K, others . Sensing the" health state" of a community. IEEE Pervasive Computing, 2011, 11(4): 36–45

80. Shetty J, Adibi J. The enron email dataset database schema and brief statistical report. Information Sciences Institute Technical Report, University of Southern California, 2004, 4(1): 120–128

81. Sapiezynski P, Stopczynski A, Lassen D D, Lehmann S. Interaction data from the copenhagen networks study.

Scientific Data, 2019, 6(1): 315

82. Panzarasa P, Opsahl T, Carley K M. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. Journal of the American Society for Information Science and Technology, 2009, 60(5): 911–932

83. Kumar S, Spezzano F, Subrahmanian V, Faloutsos C. Edge weight prediction in weighted signed networks. In: IEEE International Conference on Data Mining. 2016, 221–230

84. Kumar S, Hooi B, Makhija D, Kumar M, Faloutsos C, Subrahmanian V. Rev2: Fraudulent user prediction in rating platforms. In: Proceedings of the International Conference on Web Search and Data Mining. 2018, 333–341

85. Leetaru K, Schrodt P A. Gdelt: Global data on events, location, and tone, 1979–2012. In: ISA Annual Convention. 2013, 1–49

86. Huang Q, Jiang J, Rao X S, Zhang C, Han Z, Zhang Z, Wang X, He Y, Xu Q, Zhao Y, others . Benchtemp: A general benchmark for evaluating temporal graph neural networks. arXiv preprint arXiv:2308.16385, 2023

87. Jin M, Li Y F, Pan S. Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs. Advances in Neural Information Processing Systems, 2022, 35: 19874–19886

88. Zhu H, Li X, Zhang P, Li G, He J, Li H, Gai K. Learning tree-based deep model for recommender systems. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2018, 1079–1088

89. Jin Y, Lee Y C, Sharma K, Ye M, Sikka K, Divakaran A, Kumar S. Predicting information pathways across online communities. arXiv preprint arXiv:2306.02259, 2023

90. Huang X, Yang Y, Wang Y, Wang C, Zhang Z, Xu J, Chen L, Vazirgiannis M. Dgraph: A large-scale financial dataset for graph anomaly detection. Advances in Neural Information Processing Systems, 2022, 35: 22765–22777

91. Bailey M A, Strezhnev A, Voeten E. Estimating dynamic state preferences from united nations voting data. Journal of Conflict Resolution, 2017, 61(2): 430–456

92. Huang S, Hitti Y, Rabusseau G, Rabbany R. Laplacian change point detection for dynamic graphs. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2020, 349–358

93. Fowler J H. Legislative cosponsorship networks in the us house and senate. Social Networks, 2006, 28(4): 454–465

94. MacDonald G K, Brauman K A, Sun S, Carlson K M, Cassidy E S, Gerber J S, West P C. Rethinking agricultural trade relationships in an era of globalization. BioScience, 2015, 65(3): 275–289

95. Béres F, Pálovics R, Oláh A, Benczúr A A. Temporal walk based centrality metric for graph streams. Applied Network Science, 2018, 3(1): 1–26

96. Leskovec J, Kleinberg J, Faloutsos C. Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2005, 177–187

97. Schäfer M, Strohmeier M, Lenders V, Martinovic I, Wilhelm M. Bringing up opensky: A large-scale ads-b sensor network for research. In: Proceedings of the International Symposium on Information Processing in Sensor Networks. 2014, 83–94

98. Gehrke J, Ginsparg P, Kleinberg J. Overview of the 2003 kdd cup. ACM SIGKDD Explorations Newsletter, 2003, 5(2): 149–151

99. Weber M, Domeniconi G, Chen J, Weidele D K I, Bellei C, Robinson T, Leiserson C E. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. arXiv preprint arXiv:1908.02591, 2019

100. Poursafaei F, Huang S, Pelrine K, Rabbany R. Towards better evaluation for dynamic link prediction. Advances in Neural Information Processing Systems, 2022, 35: 32928–32941

101. Pan S J, Yang Q. A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 2009, 22(10): 1345–1359

102. Neyshabur B, Sedghi H, Zhang C. What is being transferred in transfer learning? Advances in Neural Information Processing Systems, 2020, 33: 512–523

103. Wang H, Mao Y, Sun J, Zhang S, Zhou D. Dynamic transfer learning across graphs. arXiv preprint arXiv:2305.00664, 2023

104. Hu Z, Dong Y, Wang K, Chang K W, Sun Y. Gpt-gnn: Generative pre-training of graph neural networks. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2020, 1857–1867

105. Qiu J, Chen Q, Dong Y, Zhang J, Yang H, Ding M, Wang K, Tang J. Gcc: Graph contrastive coding for graph neural network pre-training. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2020, 1150–1160

106. Chen K J, Zhang J, Jiang L, Wang Y, Dai Y. Pre-training on dynamic graph neural networks. Neurocomputing, 2022, 500: 679–687

107. Bei Y, Xu H, Zhou S, Chi H, Zhang M, Li Z, Bu J. Cpdg: A contrastive pre-training method for dynamic graph neural networks. arXiv preprint arXiv:2307.02813, 2023

108. Sharma K, Raghavendra M, Lee Y C, Kumar S, others . Representation learning in continuous-time dynamic signed networks. arXiv e-prints, 2022, arXiv–2207

109. Dai E, Wang S. Towards self-explainable graph neural network. In: Proceedings of the ACM International Conference on Information and Knowledge Management. 2021, 302–311

110. Xie J, Liu Y, Shen Y. Explaining dynamic graph neural networks via relevance back-propagation. arXiv preprint arXiv:2207.11175, 2022

111. Zheng K, Ma B, Chen B. Dynbraingnn: Towards spatio-temporal interpretable graph neural network based on dynamic brain connectome for psychiatric diagnosis. In: International Workshop on Machine Learning in Medical Imaging. 2023, 164–173

112. Brown T, Mann B, Ryder N, Subbiah M, Kaplan J D, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D. Language models are few-shot learners. In: Larochelle H, Ranzato M, Hadsell R, Balcan M, Lin H, eds, Advances in Neural Information Processing Systems. 2020, 1877–1901

113. Zhuang Y, Yu Y, Wang K, Sun H, Zhang C. Toolqa: A dataset for llm question answering with external tools. Advances in Neural Information Processing Systems, 2024, 36

114. Mendonca J, Pereira P, Moniz H, Carvalho J P, Lavie A, Trancoso I. Simple llm prompting is state-of-the-art for robust and multilingual dialogue evaluation. In: Proceedings of the Dialog System Technology Challenge. 2023, 133–143

115. Zhang Z, Wang X, Zhang Z, Li H, Qin Y, Wu S, Zhu W. Llm4dyg: Can large language models solve problems on dynamic graphs? arXiv preprint arXiv:2310.17110, 2023

116. Tang J, Yang Y, Wei W, Shi L, Su L, Cheng S, Yin D, Huang C. Graphgpt: Graph instruction tuning for large language models. arXiv preprint arXiv:2310.13023, 2023

Yanping Zheng is a Ph.D. candidate at Gaoling School of Artificial Intelligence, Renmin University of China, advised by Professor Zhewei Wei. She received her master's degree of engineering from Beijing Technology and Business University in 2020. Her research focuses on graph learning algorithms. She is particularly interested in efficient algorithms on Graph Neural Networks, Dynamic Graph Representation Learning.

Lu Yi is currently a Ph.D. student at Gaoling School of Artificial Intelligence, Renmin University of China, advised by Professor Zhewei Wei. She received her B.E. degree in Computer Science and Technology at School of Computer Science, Beijing University of Posts and Telecommunications in June 2022. Her research lie in the field of graph-related machine learning and efficient graph algorithm.
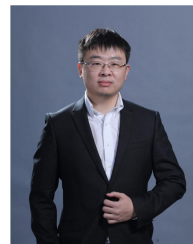
Zhewei Wei is currently a Professor at Gaoling School of Artificial Intelligence, Renmin University of China. He obtained his Ph.D. degree at Department of Computer Science and Engineering, HKUST in 2012. He received the B.Sc. degree in the School of Mathematical Sciences at Peking University in 2008. His research interests include graph algorithms, massive data algorithms, and streaming algorithms. He was the Proceeding Chair of SIGMOD/PODS2020 and ICDT2021, the Area Chair of ICML 2022/2023, NeurIPS 2022/2023, ICLR 2023, WWW 2023. He is also the PC member of various top conferences, such as VLDB, KDD, ICDE, ICML and NeurIPS.