IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 February 2024, accepted 10 March 2024, date of publication 18 March 2024, date of current version 27 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3378111

SURVEY

Dynamic Graph Representation Learning With Neural Networks: A Survey

LESHANSHUI YANG[1,3], CLÉMENT CHATELAIN[2], AND SÉBASTIEN ADAM[3]
[1]Saagie, 76140 Le Petit-Quevilly, France
[2]LITIS UR 4108, INSA Rouen Normandie, 76000 Rouen, France
[3]LITIS UR 4108, Université de Rouen Normandie, 76000 Rouen, France

Corresponding author: Leshanshui Yang (leshanshui.yang@saagie.com)

This work was supported by the ANR Labcom Lisa under Grant ANR-20-LCV1-0009.

ABSTRACT In recent years, Dynamic Graph (DG) representations have been increasingly used for modeling dynamic systems due to their ability to integrate both topological and temporal information in a compact representation. Dynamic graphs efficiently handle applications such as social network prediction, recommender systems, traffic forecasting, or electroencephalography analysis, which cannot be addressed using standard numerical representations. As a direct consequence, dynamic graph learning has emerged as a new machine learning problem, combining challenges from both sequential/temporal data processing and static graph learning. In this research area, the Dynamic Graph Neural Network (DGNN) has become the state-of-the-art approach and a plethora of models have been proposed in the very recent years. This paper aims to provide a review of the problems and models related to dynamic graph learning. The various dynamic graph supervised learning settings are analyzed and discussed. We identify the similarities and differences between existing models concerning the way time information is modeled. Finally, we provide guidelines for DGNN design and optimization, and review public datasets for evaluating model performance on various tasks, along with the corresponding publications.

INDEX TERMS Dynamic graph representation learning, dynamic graph embedding, dynamic graphs, dynamic graph neural networks.

I. INTRODUCTION

Graphs are data structures used for representing both attributed entities (the vertices of the graph) and relational information between them (the edges of the graph) in a single and compact formalism. They are powerful and versatile, capable of modeling irregular structures such as skeletons, molecules, transport systems, knowledge graphs, or social networks, across different application domains such as chemistry, biology, or finance. This expressive power of graphs explains why they have been used extensively in the last few years for real-world applications.

The ability of a system to learn and reason from graphs is important for tasks such as classification, regression, and clustering. Unlike traditional methods that rely on hand-crafted features, graph representation learning learns how to encode attributes and structural information on graphs into low-dimensional vectors. This paradigm shift has led to state-of-the-art results in many domains, such as chemical synthesis, recommender systems, and social network analysis. [1]

In the machine learning literature, most of the existing contributions focus on static graphs where the node set, the edge set, and the nodes/edge attributes do not evolve with time. Yet, for some real-world applications such as traffic flow forecasting, rumor detection, or link prediction in a recommender system, graphs are asked to handle time-varying topology and/or attributes, in order to model dynamic systems. Several terms are used in the literature to refer to graphs in which the structure and the attributes of nodes/edges evolve. Dynamic graphs [2], [3], temporal graphs [4], [5], [6], time (or temporally) evolving graphs [7],

The associate editor coordinating the review of this manuscript and approving it for publication was Kumaradevan Punithakumar.

[8], [9], [10], time-varying graphs [11], [12], time-dependent graphs [13], [14], or temporal networks [15], [16], [17], [18] are examples of such terms which refer to conceptual variants describing the same principles. This multiplicity of terms can be explained by the diversity of the scientific communities interested in this kind of models but also by the youth of the field. It illustrates the need for precise definitions and clear taxonomies of problems and models, which is one of the contributions of this paper. In the following, we choose to use the more general term Dynamic Graph (DG), especially in light of Hamilton's remark in "*Graph Representation Learning*" [1] that "*the term 'graph' appears to be more prevalent in machine learning community, perhaps in some part due to the terminological clash with 'neural networks.'*...".

As a direct consequence of the emergence of dynamic graph representations, dynamic graph learning emerged as a new machine learning problem, combining challenges from both sequential/temporal data processing and static graph learning.

When learning on sequential data, the fundamental challenge is to capture the dependencies between the different entities of a sequence. In this domain, the original concept of recurrence, mainly instantiated by LSTM [19], has been gradually replaced in recent years by convolutional architectures [20], [21], which offer better parallelization capabilities during learning. More recently, sequence-to-sequence models based on the encoder/decoder framework have been proposed [22], allowing to deal with desynchronized input and output signals. These models, such as the famous Transformer [23], rely on the intensive use of the attention mechanism [24], [25], [26].

When learning on static graphs, the main challenge is to overcome the permutation invariance/equivariance property inherent to the absence of node ordering in graph representations. Benefiting from the rapid development of the learning algorithms and hardware, geometric deep learning (or graph neural networks) [27] has recently emerged, which can iterate over the given inputs and outputs to learn the parameters that encode the input features into low-dimensional informative vectors. To solve the graph representation learning problem, node-based message-passing mechanisms based on graph structure have led to the first generation of Graph Neural Networks (GNNs), called Message Passing Neural Networks (MPNNs) [28]. As convolutions on images, these models propagate the features of each node to neighboring nodes using trainable weights that can be shared with respect to the distance between nodes (Chebnet) [29], to the features of the connected nodes (GAT) [30] and/or to edge features (k-GNN) [227]. Given the maturity of such models and their applicability for large sparse graphs, they have been applied with success on many downstream tasks. This maturity also explains the existence of some exhaustive reviews and comparative studies, such as in [31], [32], [33], and [34] to cite a few. One can note that, despite these successes, it has been shown that MPNNs are not powerful enough [33]. That is why machine learning on graphs is still a very active field, trying to improve the expressive power of GNNs [35], [36], [37].

Compared to neural networks applied for learning on sequences and on static graphs, Dynamic Graph Neural Network (DGNN) is a much more recent field. Prior to the development of DGNNs, algorithms for DG predictive tasks relied on traditional graph-theoretic methods, such as those based on non-negative tensor factorization [38]. Until recently, such methods and algorithms are still being refined and used for tasks such as embedding and clustering on dynamic graphs [40], [41]. As a relatively young technique, to the best of our knowledge, the founding DGNN models are from 2018 [42] and 2019 [43] for respectively the discrete and the continuous cases. These papers have been at the root of a "zoo" of methods proposed by various scientific communities, with various terminologies, various learning settings, and various application domains. Up until today, many novel state-of-the-art approaches have been proposed within the DGNN field, such as GraphMixer [129], which explores the combination of the MLP-Mixer structure [39] with DGNNs, the Euler framework [89] for distributed systems; and architectures exploring the combination of Transformers and Dynamic Graph Neural Networks [73], [122].

In order to structure the domain, some state-of-the-art papers have been published recently [44], [45], [46], [47]. Without emphasizing the DGNNs, these papers give a good overview of many machine learning issues linked to dynamic graphs and describe many existing models. However, several limitations prevent the reader from understanding more complete and up-to-date techniques. For example, these surveys do not address the recent use of transformer models [23] in dynamic graphs; Surveys [45], [46], [47] did not discuss models for spatio-temporal graphs, which are introduced in detail in this survey. More importantly, this survey attempts to provide perspectives and taxonomies of DGNNs that are more accessible to machine learners and graph theorists, including detailed discussions of design and optimization strategies for DGNNs, along with a synthesis of datasets and relevant publications.

The main objective of this review is to extend the existing studies mentioned above, by focusing on **dynamic graph supervised learning using neural networks**. It is addressed to the audience with fundamental knowledge of neural networks and static graph learning. Three main contributions can be highlighted. The first one consists in **clarifying and categorizing the different dynamic graph learning contexts** that are encountered in the literature. These contexts are distinguished according to the type of input DGs (discrete vs. continuous, edge-evolving vs. node-evolving vs attributes-evolving, homogeneous vs. heterogeneous) but also according to the learning setting (transductive vs. inductive). The second contribution is an **exhaustive review of**

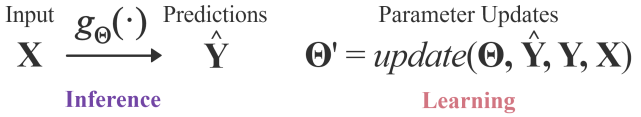


FIGURE 1. Left: inference phase for making predictions $\hat{\mathbf{Y}}$ on given data $\mathbf{X}$. Right: learning phase for updating the parameters Θ of the predictor g.

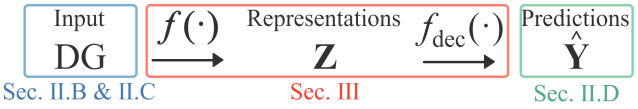


FIGURE 2. Encoder/decoder model applied on dynamic graphs: the encoding consists in computing $\mathbf{Z} = f(\text{DG})$, where DG is a dynamic graph (including both topology and attributes), $f(\cdot)$ is a parameterized statistical model (typically a neural network with learnable parameters), and $\mathbf{Z}$ is the encoded tensor representation of DG. The decoder $f_{dec}(\cdot)$ takes as input the representations $\mathbf{Z}$ to get the predictions $\hat{\mathbf{Y}}$.

existing DGNN models, including the most recent ones. For this review, we choose to categorize models into six groups, according to the strategy used to incorporate time information in the model, which is the main challenge for the application of neural networks on DGs. Based on this categorization, and using the taxonomy of contexts mentioned above, the third contribution is to provide some **general guidelines for designing, optimizing, and evaluating DGNNs**.

The remainder of this paper is structured as follows. Section II relates to the first contribution, by considering the inputs, the outputs, and the learning settings that can be encountered when learning on dynamic graphs. Section III reviews existing Dynamic Graph Neural Networks (DGNNs) and compares them according to their temporal information processing. Finally, section IV brings forward the guidelines for designing and optimizing DGNNs, along with a synthesis of common datasets and relevant publications for evaluating them.

II. REPRESENTATION LEARNING OF DYNAMIC GRAPHS

In this section, we define important concepts about representation learning on dynamic graphs, providing the necessary material for understanding the review of embedding approaches presented in section III. The section is structured as follows: after positioning the context of representation learning in subsection II-A, we give useful definitions about the representation of static graphs and dynamic graphs in subsections II-B and II-C. We then discuss the possible outputs of machine learning models operating on dynamic graphs and the transductive/inductive nature of learning tasks in subsections II-D and II-E. Finally, we present examples of applications in each learning setting in subsection II-F.

A. REPRESENTATION LEARNING

Regardless of the data representation, the goal of supervised learning methods is to build a parameterized statistical model or predictor g_Θ that maps between an input space $\mathcal{X}$ and an output space $\mathcal{Y}$ (see Fig. 1). During the *learning* phase, the training of the predictor g_Θ consists in updating its parameters Θ using a dataset $(\mathbf{X}, \mathbf{Y})$ of couples (x, y) with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. The update is computed by a minimization of the loss between the predictions $\hat{\mathbf{Y}}$ and the ground truth $\mathbf{Y}$.

When learning on structured data such as sequences, images, or graphs, information varies according to the position in a structure. This structure is generally a grid that may have one or multiple dimensions: 1D for text (i.e. the position of the word) or speech (i.e. the time), 2D position for images, etc.

When the size of the structure mentioned above may vary, recent models frequently follow the encoder/decoder principle, where a variable-length input signal is encoded into a latent representation, which is then used by a decoder to compute the output signal for the downstream task (see Fig. 2 in the case of dynamic graphs). The latent representation allows processing variable-size input and output signals that are not necessarily synchronized. Encoder/decoder models are of great interest in many sequence-to-sequence problems involving text, images, or speech. In this context, learning the latent representation (also known as *embedding*) is called *representation learning*.

In the case of dynamic graphs, the information varies according to both the position in the graph and the time. Moreover, the structure of the graph itself can also evolve over time. Before diving into the representation learning of dynamic graphs, the following subsections give the necessary definitions concerning the inputs, the outputs, and the learning tasks.

B. STATIC GRAPH MODELING

A static graph G can be represented topologically by a tuple (V, E) where V is the node set of G and E is the edge set of G. The connectivity information is usually represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$. In this matrix, $\mathbf{A}(u, v) = 1$ if there is an edge between node u and v, $\mathbf{A}(u, v) = 0$ otherwise. $\mathbf{A}$ is symmetric in an undirected graph, while in the case of directed graphs $\mathbf{A}$ is not necessarily symmetric.

Nodes usually have attributes represented by a feature matrix $\mathbf{X}_V \in \mathbb{R}^{|V|\times d_V}$, where d_V is the length of the attribute vector of a single node. Similarly, edges may have attributes (such as weights, directions, etc.) which can also be represented by a matrix $\mathbf{X}_E \in \mathbb{R}^{|E|\times d_E}$.

In the case of weighted graphs, the values in the matrix $\mathbf{A}$ are the weights of each edge instead of 1 which is denoted as:

$$\mathbf{A}_{u,v} = \begin{cases} w_{u,v} & \text{if } (u, v) \in E \\ 0 & \text{otherwise.} \end{cases}$$

For some specific applications, both nodes and edges can be of different types. For example, in recommender systems, nodes can usually be mapped into two types, item, and user, and can have feature matrices of different sizes and contents. In such cases, we extend the notation of the graph to $G = (V, E, \phi, \psi)$ with the type mapping functions $\phi : V \to O$ and $\psi : E \to R$, where $|O|$ denotes the possible types of

TABLE 1. Degree of Dynamism discussed in the article.

		Node Set	
		Constant	Variant
Edge Set	Constant	$fix_{V,E}$	N/A
	Variant	fix_V	$vary$

nodes and $|R|$ denotes the possible types of edges [48]. When $|O| = 1$ and $|R| = 1$, nodes and edges are of a single type, this graph is called homogeneous. In contrast, in a heterogeneous graph, $|O| + |R| > 2$ and each type of node or edge could have its own number of feature dimensions [48], [49].

Figure 3 illustrates the difference between homogeneous and heterogeneous graphs.

C. DYNAMIC GRAPH MODELING

In this subsection, we first introduce the definition of a dynamic graph and the concept of Degree of Dynamism. Then, we provide the main existing formalisms for representing DG, synthesizing previous studies that focus on subparts of these representations [13], [15], [45], [50], [51], [52]. We also introduce in this part a representation called Equivalent-Static-Graph which consists in modeling a DG using a static graph. The different models are compared in Fig. 3.

Definition 1 (Dynamic Graph): A dynamic graph is a graph whose topology and/or attributes change over time.

To cover this broad research area, the definition of dynamic graphs that we propose in Def. 1 is highly generic, intending to include, as much as possible, the relevant models in the existing literature. According to this definition, both structure and attributes may change over time in a dynamic graph. Edges and/or nodes may be added or deleted and their attributes may change. Thus, this definition covers different configurations. In order to distinguish between them, we propose in this paper the concept of *degree of dynamism* defined as follows:

Definition 2 (Degree of Dynamism (Node-Centric)): The *degree of dynamism* of a DG describes whether the topology, i.e. the edge set E and the node set V, vary or not. Theoretically, there are 4 possible situations: (1) Both V and E are invariant, denoted as $fix_{V,E}$. This situation corresponds to DG called "Spatial-Temporal Graphs" or "Spatio-Temporal Graphs" (STGs) in the literature. (2) V is invariant but the set of edges change, denoted as fix_V. (3) The node set and the edge set are both changing, denoted as *vary*. (4) The set of edges is constant but the set of nodes changes. Since an edge is defined by a tuple of nodes, this situation is meaningless.

Table 1 illustrates these different configurations which will be discussed throughout the paper.

Irrespective of the degree of dynamism, three main representation formalisms can be found to describe a DG. They are listed below.

1) CONTINUOUS TIME DYNAMIC GRAPHS

To preserve accurate time information, Continuous Time Dynamic Graphs (CTDGs) use a set of events to represent dynamic graphs. As mentioned in the review [45], existing works mainly focus on the dynamics of edges and outline three typical representation methods that represent events with index $i = 1, 2, \ldots$ by giving a pair of nodes (u_i, v_i) and time t_i:

Contact-Sequence methods [15], [45] represent the instantaneous interaction between two nodes (u, v) at time t:

$$Contact - Sequence = \{(u_i, v_i, t_i)\} \tag{1}$$

Event-Based methods [45], [53] represent edges with a time t_i and a duration Δ_i. They are similar to the *Interval-Graph* defined in [15]. The difference is that *Interval-Graph* uses a set T_e of start and end times (t_i, t_i') to represent all active times of the edge, rather than the duration of each interaction in *Event-Based*.

$$Event - Based = \{(u_i, v_i, t_i, \Delta_i)\} \tag{2}$$

$$Interval - Graph = \{(u_i, v_i, T_e)\}; \quad T_e = \left((t_1, t_1'), (t_2, t_2'), \ldots\right) \tag{3}$$

Graph-Stream methods [45] are often used on massive graphs [54], [55]. They focus on edges' addition ($\delta_i = 1$) or deletion ($\delta_i = -1$).

$$Graph - Stream = \{(u_i, v_i, t_i, \delta_i)\} \tag{4}$$

2) DISCRETE TIME DYNAMIC GRAPHS

Discrete-time dynamic graphs (DTDGs) can be viewed as a sequence of T static graphs as shown in (5). They are snapshots of the dynamic graph at different moments or time-windows. DTDGs can be obtained by periodically taking snapshots of CTDGs on the time axis [45], [50].

$$DTDG = (G^1, G^2, \ldots, G^T) \tag{5}$$

3) EQUIVALENT STATIC GRAPHS

Representations of this category consist in constructing a single static graph, called Equivalent Static Graph (*ESG*), for representing a dynamic graph. Several methods for constructing *ESG* have been proposed in recent years. We divide them into two categories: *edge-oriented* and *node-oriented ESG*.

Edge-oriented ESG aggregate graph sequences into a static graph with time information encoded as sequences of attributes [5], [15], [16] as shown in Fig. 3 (right top). Such representations are also called *time-then-graph* representations [52].

Node-oriented ESG build copies of vertices at each moment of their occurrence and define how the nodes are connected between timestamps/occurrences [5], [56], [57], [58]. A simple example is shown in Fig. 3 (right bottom), further details are discussed in subsection III-B.

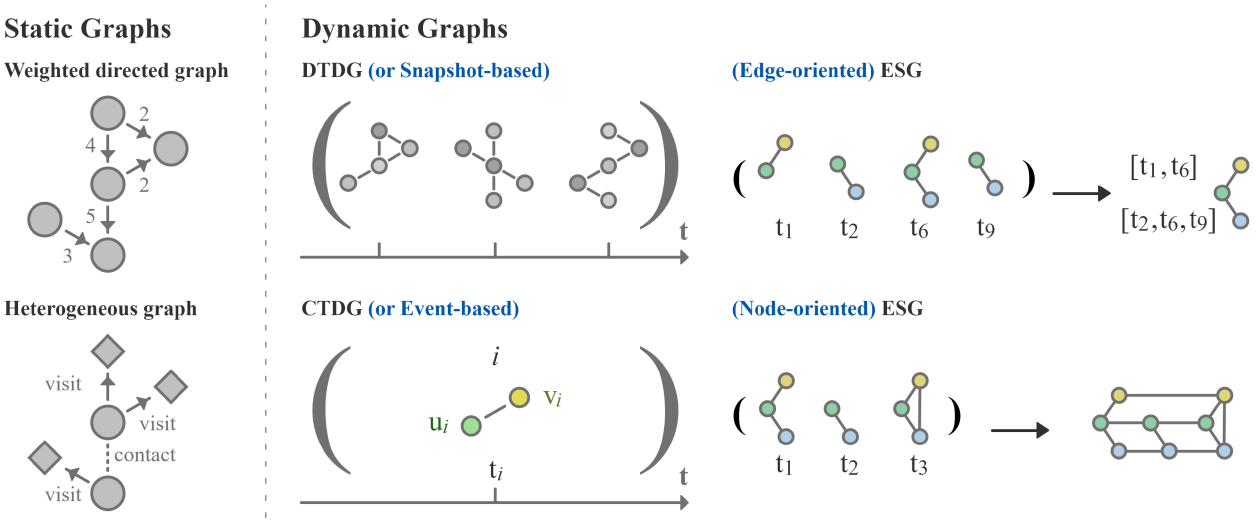


FIGURE 3. Left: Common static graph representations. The edges in a weighted directed graph have directions and weights, which are frequently used when modeling email networks, citation networks, etc. The nodes and edges in a heterogeneous graph can have multiple possible types i.e., recommendation systems and knowledge graphs. Middle: Discrete Time Dynamic Graph (DTDG) represented by snapshots, and Continuous Time Dynamic Graph (CTDG) represented by events, the example in the figure is the "Contact Sequence" case. Right: Equivalent Static Graphs (ESGs) represented by edges and nodes.

A major interest of ESG representations is that they make static graph algorithms available for learning on dynamic graphs.

D. DYNAMIC GRAPH OUTPUT GRANULARITY

As said before, supervised learning aims to learn a mapping function between an input space $\mathcal{X}$ and an output space $\mathcal{Y}$. The previous subsection introduces how a DG $x \in \mathcal{X}$ can be represented as input of a model. This subsection now discusses the output, i.e. the space $\mathcal{Y}$.

Since a dynamic graph involves concepts from both graph and temporal/sequential data, both aspects must be considered to define the space $\mathcal{Y}$.

Sequential data are generally made of feature vectors of length d that evolve over time, denoted as $\mathbf{X} \in \mathbb{R}^{T \times d}$ for a sequence of T time steps. Depending on the task, the output Y can be either a label (sequence classification problems), a value (sequence regression problem), a sequence of labels of the same size as the input size (sequence labeling), or a sequence of labels of a different size from that of the input (sequence-to-sequence problem).

In static graph learning, the inputs have topological information in addition to the features $\mathbf{X}_V$ and $\mathbf{X}_E$. As for sequences, the outputs $\mathbf{Y}$ can be local (one label per node or per edge) or global (one label per subgraph or for the entire graph).

As a consequence, the output granularity when considering DG can be temporally timestep-level or aggregated, and topologically local or global. The output contents can be categorical labels for classification or numerical values for regression.

E. TRANSDUCTIVE/INDUCTIVE ON DYNAMIC GRAPHS

When learning on static graphs, transductive and inductive tasks are frequently distinguished. Transductive tasks consist in taking decision for a set $V_{inference}$ of unlabeled nodes, the model being learned on a set of labeled nodes $V_{learning}$ of the same graph. In such a situation, the features and the neighborhood of nodes in $V_{inference}$ can be exploited by the learning algorithm in an unsupervised way (i.e., learn and find patterns with unlabeled data). For this reason, this situation is also called semi-supervised learning on graphs [59]. In contrast, an inductive task is the case when there are nodes in the $V_{inference}$ that were not seen during the learning phase [9]. This situation typically occurs when learning and inference are performed on different static graphs.

In the case of dynamic graphs, the evolving nature of both V and E brings more possible scenarios when considering the transductive/inductive nature of the tasks. Various configurations should be defined, considering the "degree of dynamism" defined in section 2. We use the term "case" to distinguish between different transductive/inductive natures of dynamic graph learning taking into account the degree of dynamism.

Specifically, we divide learning on dynamic graphs into five cases:

- *Trans-fix*$_{V,E}$ (1): in this case, the topology of the DG is fixed, V and E are therefore the same on the learning and inference sets.
- *Trans-fix*$_V$ (2): in this case, the learning and inference node sets are fixed and equal but the edge set evolves, which requires taking into account the evolving connectivity between nodes on the graph.
- *Trans-vary* (3): in this case, $V^t_{learning}$ and $V^t_{inference}$ may evolve over time, but the presence of each node to be predicted in the test set are already determined in the learning phase: $\forall v \in V_{inference}, v \in DG_{learning}$.
- *Ind*$_V$ (4): this case refers to node-level inductive tasks, where the learning and inference are on the same DG, but $\exists v \in V_{inference}, v \notin DG_{learning}$. Although the label

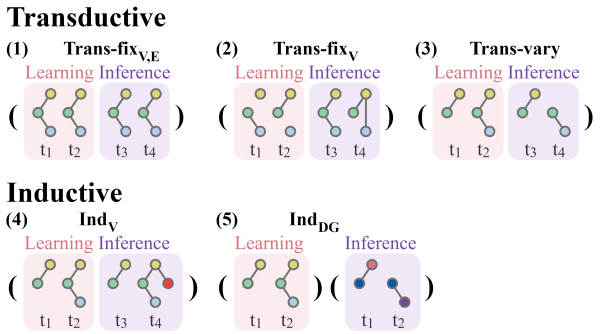


FIGURE 4. The transductive and inductive cases in dynamic graph learning under discrete time: (1) denotes the case where both the node set and edge set are fixed in learning and inference. (2) denotes the case where only the node set is fixed for learning and inference. (3) denotes the case where the node set changes but no unseen node appears for inference. (4) denotes the inductive case where the train and test are on the same DG but there are new nodes for inference. (5) denotes the inductive case where there are new DGs for inference.

and historical attributes from the learning phase can be reused on the inference set, a major challenge is to handle the unseen nodes and the uncertain number of nodes.

- Ind_{DG} (5): this case refers to DG-level inductive learning, where learning and inference are performed on different DGs. The statistical model needs to handle the complete unseen dynamic graphs.

These 5 cases are illustrated in Fig. 4. In all cases, the learning of vector representations of nodes that change over time is the basis for dynamic graph learning. The representation learning of edges and (sub)graphs is optional, depending on the specific predictive task, because they can be obtained by operations on the representations of a pair of nodes or multiple nodes.

F. DYNAMIC GRAPH PREDICTIVE TASKS

In the previous subsections, we have investigated dynamic graph learning specificities related to the representations of the input (discrete vs. continuous time), the granularity of the output (local vs. global, timestep-based vs. aggregated), and different learning cases. In this section, we categorize existing contributions in the literature according to these four criteria.

Table 2 gives a synthetic view of this categorization with an emphasis on the targeted applications and on the metrics used for assessing models' performance.

As one can see in this table, in **Discrete Time Transductive Tasks**, related applications usually concern relatively stationary topologies, i.e. $Trans\text{-}fix_{V,E}$, such as human body structure or geographic connectivity. Some typical node-level/local tasks predict attributes for the next time step(s) based on past time step(s), such as traffic flow [51], [60], [61], [62], [63], number of infectious disease cases [64], [65], [66], number of crimes [67] and crop yields [68]. Graph-level/global tasks either retrieve the class of each snapshot like sleep stage classification [69] or output a prediction for the entire DG such as the emotion of a skeletal STG [70]. When the DG has a fixed set of nodes with evolving edges, i.e. $Trans\text{-}fix_{V}$, such cases can be employed for modeling the connectivity in the telecommunication network [10] or contact of individuals in a conference [18]. In recent research, anomaly detection on dynamic graphs has gained prominence. The main goal is to detect anomalous graph objects [71]. Most contemporary research using neural networks focuses on DTDGs with topological changes, i.e. the *Trans-vary* case. In this context, 1% to 10% of non-existent edges are introduced into each snapshot, and the goal is to detect them with edge classification. This task has been extensively adopted in various domains, such as blockchain networks [72], [73], [74], brain networks [72], email networks [73], [75], [76], and for identifying attacks on enterprise networks [74].

In **Discrete Time Inductive Tasks**, i.e. when unseen nodes need to be predicted in discrete time i.e. Ind_{V}, some typical tasks concerns node classification [4] or link prediction [4], [18], [77] for future snapshots in social networks. An example of Ind_{DG} with graph-level output is classifying real and fake news based on the snapshots of its propagation tree on the social network [78], [79].

Dynamic graphs in **Continuous Time** are widely used to model massive dynamic graphs that frequently have new events, such as recommendation systems [80], [81], [82] or social networks in the transductive cases [83], [84] or inductive cases [2], [6], [17], [43]. Local tasks predict the properties and interactions of seen or unseen nodes, under transductive and inductive cases, respectively. Since CTDGs have no access to global/entire graph information under their minimum time unit, the global timestep level label is meaningless under continuous time. However, global aggregated tasks can be implemented by aggregating nodes of different time steps, such as rumor detection in continuous time [8]. Since CTDG can be transformed to DTDG by periodically taking snapshots [45], [50], the above tasks can also be considered as tasks under dynamic time with a lower time resolution.

To evaluate the performance of a statistical model on a given task, traditional machine learning metrics are employed as shown in table 2. When the output predictions are discrete values, i.e. for **classification** task [2], [4], [6], [8], [43], common metrics include accuracy, precision, recall, F1, and area under the receiver operating characteristic (AUROC). When the output values are continuous values, i.e. for **regression** task [51], [60], [61], common metrics are mean absolute (percentage) error, root mean square (log) error, and correlation. **Node ranking** tasks [81], [82] predict a score for each node and then sort them. These tasks can be evaluated by the reciprocal rank, recall@N, cumulative gain, and their variants. Note that dynamic tasks are generally evaluated using static metrics computed along the time axis.

III. DYNAMIC GRAPH EMBEDDING WITH NEURAL NETWORKS

In the previous section, we have introduced various dynamic graph predictive task settings and we have categorized

TABLE 2. Common dynamic graph predictive tasks and related applications categorized by their input time granularity, transductive/inductive cases, and label shapes. "Clf." indicates classification and "Reg." indicates regression. Social networks also contain contact network and citation network, whether it is heterogeneous or not. Where "DT" refers to "Discrete time" and "CT" refers to "Continuous time".

Input time	Case	Label object	Label time	Application	Clf.	Reg.	Metrics	Publications
DT	*Trans-fix*$_{V,E}$	Global	Aggregated	Video clf.	✓		Acc., AUROC	[106], [107]
	Trans-fix$_{V,E}$	Global	Aggregated	EEG-based task and gender clf.	✓		Acc., AUROC	[108]
	Trans-fix$_{V,E}$	Global	Aggregated	Skeleton-based emotion clf.	✓		Acc.	[70]
	Trans-fix$_{V,E}$	Global	Timestep	EEG-based sleep stage clf.	✓		Acc., F1	[69]
	Trans-fix$_{V,E}$	Local	Both	Crop yield prediction		✓	RMSE, Corr., R^2	[68]
	Trans-fix$_{V,E}$	Local	Both	Traffic flow forecasting		✓	MAE, MAPE, RMSE	[51], [60]–[63]
	Trans-fix$_{V,E}$	Local	Both	Influenza forecasting		✓	RMSE, MAE, Corr	[64]–[66]
	Trans-fix$_{V,E}$	Local	Both	Crime prediction	✓	✓	F1	[67]
	Trans-fix$_{V}$	Local	Both	Social net. link prediction	✓		AUC, Pr., Rec., F1	[4], [77], [105], [110]
	Trans-fix$_{V}$	Local	Both	Social net. node clf.	✓		AUC, F1	[4], [18]
	Trans-fix$_{V}$	Local	Both	Path (edge set) availability clf.	✓		Pr., Rec., F1	[10]
	Trans-vary	Local	Aggregated	Social net. node clf.	✓		Acc., F1	[88], [90]
	Trans-vary	Local	Timestep	Anomalous edges clf.	✓		AUC	[71]–[76]
DT	*Ind*$_{V}$	Local	Both	Social net. link prediction	✓		AUC, Pr., Rec., F1	[3], [115]
	Ind$_{V}$	Local	Timestep	Social net. attribute reg.		✓	RMSE, MAE	[203], [204]
	Ind$_{V}$	Local	Timestep	Agriculture trading reg.		✓	MAE	[205]
	Ind$_{DG}$	Global	Aggregated	Rumor detection	✓		Acc., Pr., Rec., F1	[78], [79]
CT	*Trans-vary*	Local	Timestep	Social net. link prediction	✓		AP, Acc.	[83], [84]
CT	*Ind*$_{DG}$	Global	Aggregated	Rumor detection	✓		Acc., F1	[8]
	Ind$_{V}$	Local	Timestep	Social net. link prediction	✓		AP, Acc.	[2], [6], [17], [43], [82]
	Ind$_{V}$	Local	Timestep	Social net. node clf.	✓		ROC AUC	[2], [6], [82]
	Ind$_{V}$	Local	Timestep	Rating prediction		✓	MAE, RMSE	[80]
	Ind$_{V}$	Local	Timestep	Event time prediction		✓	MAE	[43]
	Ind$_{V}$	Local	Timestep	Item recommendation		✓	Recall@10, MRR	[81], [82]

literature contributions according to these settings. In this section, we now take the model point of view, by describing how DGNNs embed dynamic graphs into informative vectors for subsequent predictions.

We first introduce the general idea of dynamic graph embedding in the context of the different learning tasks mentioned in subsection III-A. We then dive into different embedding approaches, by categorizing them according to the strategy used for handling both temporal and structural information. Finally, we present the methods for handling heterogeneous dynamic graphs.

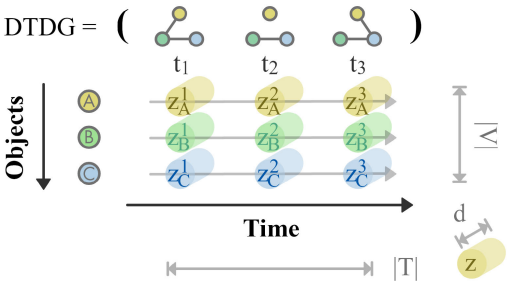


FIGURE 5. The most fine-grained node embedding $\mathbf{Z} \in \mathbb{R}^{|V|\times|T|\times d}$, where $|V|$ is the number of nodes, $|T|$ is the number of timesteps, and d is the dimension of embedding z_v^t of a single node v at a single timestep t.

A. DYNAMIC GRAPH EMBEDDING

From an encoder-decoder perspective, a deep learning statistical model first maps the original input into an embedding vector/matrix/tensor denoted as $\mathbf{Z}$, and then exploits $\mathbf{Z}$ to predict an output [44], [85].

When learning on graphs, embedding can be performed either at node/edge-level or at (sub)graph-level [46], [86]. Node-level embedding benefits a wide range of node-related tasks and allows more complete input information to be retained for later computation [86].

In the same way, time-step level embedding retains more information than time-aggregated embedding as when learning on sequential data [87].

As a consequence, embedding a dynamic graph at its finest granularity consists in computing a d-dimensional vector representation $\mathbf{z}_v^t \in \mathbb{R}^d$ for each node $v \in V$, at all time steps $t \in T$. In this case, the embedding of the dynamic graph is given by $\mathbf{Z} \in \mathbb{R}^{|V|\times|T|\times d}$, as shown in Fig. 5.

However, the different input time granularity and learning settings mentioned in the previous section do not always enable such an "ideal" embedding $\mathbf{Z}$. In this subsection, we generalize the practicable embeddings under these different settings as shown in Fig. 6.

For discrete time transductive settings, when the node set is constant, (i.e. for cases (1) and (2) of Fig. 4), the input nodes can be encoded at the finest granularity $\mathbf{Z} \in \mathbb{R}^{|V|\times|T|\times d}$ since all the nodes are known during learning, as shown in Fig. 5.

When the DTDG node set changes across snapshots in a transductive task (i.e. for case (3)), the nodes can still be encoded in the shape of $|V| \times |T| \times d$, where $|V|$ denotes the cardinal of the universal node set [88], [89], by filling the missing values with $\mathbf{0}$ vectors [90] or the latest updated node embedding [89]. An example of filling the missing values with $\mathbf{0}$ vectors [90] is shown in Fig. 6 (A) for node C at t_1 and node A at t_3.

For discrete time inductive settings (i.e., for cases (4) and (5)) the predictor cannot determine the existence of a node until its first appearance. This case is illustrated

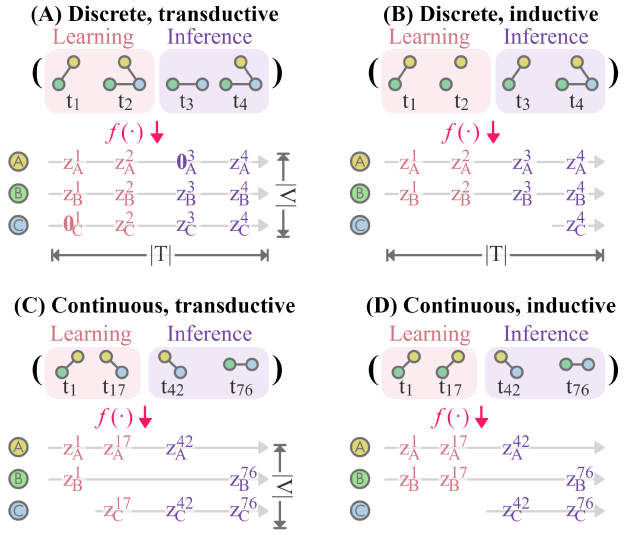


FIGURE 6. The most fine-grained embedding available under different dynamic graph learning settings. The embedding of node v at moment t is denoted $\mathbf{z}_v^t$, and the timeline on the right side of each node indicates its available embeddings. In case (A) Discrete, transductive setting, since the total number of nodes $|V|$ and time steps $|T|$ are known, the entire embedding of a dynamic graph can thus be represented by a tensor of shape $|V| \times |T| \times d$. In case (B), the total number of nodes is unknown in the learning phase due to the inductive setting. In continuous setting (C) or (D), typically, the embedding of a node is updated only when it has edges with other nodes, e.g., node A at timestamps t_1, t_{17}, and t_{42}.

in Fig. 6 (B) where the predictor cannot determine the existence of node C until it appears at t_4. Hence, $|V^t|$ can vary at each timestep t. Therefore, the embedding of nodes in the inference set cannot be represented in the shape of $|V| \times |T| \times d$. In this situation, existing approaches use a list of hidden state matrices of different sizes to store the representations of all accessible time steps for each node seen [8], [78], [79], e.g. $\mathbf{Z} = \{\mathbf{Z}^1, \mathbf{Z}^2, \ldots, \mathbf{Z}^T\}$ with $\mathbf{Z}^t \in \mathbb{R}^{|V^t| \times d}$ for $t \in \{1, \ldots, T\}$.

In continuous time, there is no longer a time grid, as shown in Fig. 6 (C & D). Therefore, there are no longer embedding updates for all nodes at each time step. Instead, when an event occurs on the CTDG, either the embeddings of the associated nodes are updated or the embedding(s) of the unseen node(s) are added [2], [6].

Having delineated the various forms of node embedding in discrete-time and continuous-time scenarios, we now explore various dynamic graph neural network (DGNN) architectures. To informatively encode dynamic graphs into tensors or a list of vectors, a DGNN must capture both the structure information and its evolution over time. Therefore, to handle topology and time respectively, DGs are often decomposed or transformed into components like equivalent static (sub)graphs [91], [92], [93], random walks [17], [18], [84], [94], [95], or sequences of matrices [69], [78], [90]. In the literature, numerous approaches have emerged by combining different encoders $f_G(\cdot)$ for static graphs with $f_T(\cdot)$ for temporal data. A plethora of graph and temporal data encoders have been at the root of the DG encoders reviewed in the section. These encoders are described in appendices A and B.

In the following subsections, we present a categorization of existing DGNN models. The taxonomy is based on the strategy used to handle both temporal and structural information. This leads to the 5 categories shown in Figure 7. A sixth category, corresponding to very recent transformer models that mix ideas from the previous ones, is then described:

1) Modeling temporal edges through node-oriented ESG, denoted as *TE* (Section III-B).
2) Sequentially encoding the hidden states, denoted as $enc(H)$ (Section III-C).
3) Sequentially encoding the DGNN parameters, denoted as $enc(\Theta)$ (Section III-D).
4) Embedding occurrence time t as edge feature of edge-oriented ESG, denoted as $emb(t)$ (Section III-E).
5) Sampling causal random walks (RWs), denoted as *CausalRW* (Section III-F).
6) Dynamic Graph Transformer, denoted as *DGT* (Section III-G).

Note that these approaches are not exclusive, i.e. they can be combined and used on the same DG.

B. TEMPORAL EDGE MODELING

Since applying convolution on a static graph is generally easier than encoding across multiple snapshots, the DG encoding problem is frequently transformed into encoding a static graph where each node is connected to itself in the adjacent snapshot [91], [92], denoted as *TE*. This approach can also be interpreted as constructing a *time-expanded graph* [5], [56], [57] or *node-oriented ESG* (see section 2.2.4) and is widely used to encode *Trans-fix*$_{V,E}$ cases, i.e., "Spatial-Temporal Graphs" or "Spatio-Temporal Graphs" (STGs). In more complex configurations, nodes are also connected with their k-hop neighbors in the adjacent snapshot(s) [64].

An example of such a strategy is shown in Fig. 8. An equivalent static graph $G' = \{V', E'_S, E'_T\}$ is obtained by modelling temporal edges [91]. G' has $|V'| = |V| \times T$ nodes and $|E'_S| = |E| \times T$ spatial edges. Depending on the modeling approach, the number of temporal edges $|E'_T| = |V| \times (T-1)$ can be greater.

Once defined the connection rule for temporal edges, the traditional convolution for static graphs is applicable on ESGs. An influential example is the **ST-GCN** module [92], which has been employed for skeleton-based action classification. To update the hidden states $\mathbf{h}$ to $\mathbf{h}'$ in an ST-GCN layer (see (6)), a typical spatial GNN aggregates the neighbor features with the msg$(\cdot)$ function, computes their weights by the w$(\cdot)$ function, and then sums them after normalization with the norm$(\cdot)$ function. Note that, unlike the neighborhood definition in static graphs, the neighborhood set N of v_i^t is defined by: (1) spatially, the shortest path distance with neighbor v_j obeys $d(v_j, v_i) \leq K$ and (2) temporally, the time difference between timestamps q and t, i.e. $|q - t|$ is not

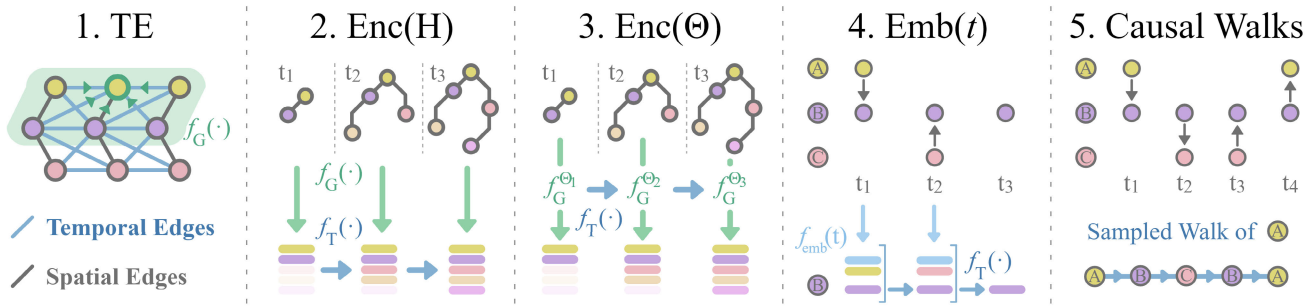


FIGURE 7. Dynamic graph neural network taxonomy on temporal information processing: 1. **Temporal Edge Modeling** models temporal edges to transform STGs into static graphs. 2. **Sequentially Encoding Hidden States H** encodes the hidden states of each snapshot across time with a temporal encoder $f_T(\cdot)$. 3. **Sequentially Encoding Parameters Θ** encodes the parameters Θ of the graph encoder $f_G(\cdot)$ across time with a temporal encoder $f_T(\cdot)$. 4. **Embedding Time** t converts time values to vectors and concatenates or adds them to the attribute vectors when encoding node B. 5. **Causal Walks** restricts the random walks on dynamic graphs by causality.

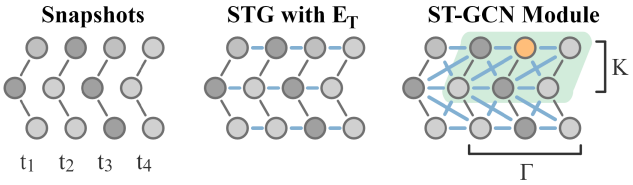


FIGURE 8. Comparing snapshot representation (left) and STG with temporal edges (middle) [91]. The edges in blue indicate temporal edges. In ST-GCN module (right) [92], temporal edges also connect nodes with their K-hop neighbors in the adjacent $\Gamma/2$ snapshot(s), thus the light green area shows the neighborhood of the orange node while applying spatial-temporal graph convolution with K=1 and Γ=3.

greater than $\lfloor \Gamma/2 \rfloor$.

$$\mathbf{h}_i^{'t} = \sum_{v_j^t \in N_{v_i^t}} norm\left(msg(\mathbf{h}_i^t, \mathbf{h}_j^t) \cdot w(\mathbf{h}_i^t, \mathbf{h}_j^t)\right)$$
$$N_{v_i^t} = \left\{ v_j^q | d(v_j^t, v_i^t) \leq K, |q - t| \leq \lfloor \Gamma/2 \rfloor \right\} \quad (6)$$

Following the methodology introduced in ST-GCN [92], **STSGCN** [96] applies the similar approach to traffic flow forecasting. It constructs temporal edges to connect each node with itself across preceding and succeeding time steps, subsequently leveraging traditional Graph Convolution layers for node encoding. Going beyond simply using the adjacency matrix of each snapshot, **TSNet** [97] proposes a sparsification network to generate sparsified subgraphs based on the learned distribution. This method samples a subset of edges for reducing the risk of overfitting. Once the sparsified subgraphs are generated, each node is then connected to its neighbors from preceding, current, and succeeding snapshots, which is referred to as ''Temporal-Structural Neighbourhood'' for subsequent graph convolution operations.

Besides using the existing graph structures, Transformers' self-attention mechanism is exploited to learn the relations between nodes over multiple time steps [98], [99], [100], [101]. An in-depth discussion about transformers-based approaches is provided in Section III-G.

C. SEQUENTIALLY ENCODING HIDDEN STATES H

In DTDGs, there are usually additions/deletions of edges or nodes [4], [10], [18], [77]. To deal with these topological changes, this category denoted as $enc(\mathbf{H})$ uses $f_G(\cdot)$ and $f_T(\cdot)$ to encode the graph and time domains alternatively. $Enc(\mathbf{H})$ is widely applied to the Trans-$fix_{V,E}$ case, i.e. STGs [51], [60], [61], [62], [63], [68], [70], [102] and the Trans-fix_V cases on DTDGs [4], [10], [18], [77], [103], [104]. $f_T(\cdot)$ either encodes each snapshot across time after $f_G(\cdot)$ encodes each snapshot [42], [88], [90], i.e., in a stacked way [45], or incorporates graph convolution when encoding each snapshot across time [42], [61], [105], i.e., in an integrated way [45].

When the input is an STG, rather than processing them as static graphs, this approach factorizes space and time and processes them differently [51], [60], [61], [65], [68], [69], [102], [103], [104], [106], [107], [108], [109]. **RSTG** [106] and **DyReG** [107] first encode each snapshot at node-level with weighted message passing as $f_G(\cdot)$, and then encode each node over time using LSTM or GRU as $f_T(\cdot)$. Both approaches are of the stacked fashion.

If the node set V_t of a DTDG is constant, then this DTDG is equivalent to an STG only with an additional change in the edge set E_t, i.e. Trans-fix_V case. Therefore, the stacked architecture mentioned in the previous paragraph is equally practicable. A typical example is **CD-GCN** [90] that concatenates the attributes and the adjacency matrix for each snapshot t to form input $\mathbf{X}^t||\mathbf{A}^t$. Each snapshot is first encoded by GCN to obtain the node-level topological hidden states $\mathbf{z}_{i,GCN}^t$, and then encoded by LSTM in the time dimension to obtain $\mathbf{z}_{i,LSTM}^t$. Finally, an MLP maps the concatenation of the hidden states and raw features to the final node-level hidden states $\mathbf{z}_i^t$ for each time step. Similar structures which stack temporal encoder $f_T(\cdot)$ after graph encoder $f_G(\cdot)$ are **STGCN** [60], **GraphSleepNet** [69], **E-LSTM-D** [110], **Graph WaveNet** [51], **GRNN** [62]. A simple example is shown in (7). As a matter of fact, they can be stacked in more complex ways, see tab 3 for more details.

$$\mathbf{Z}^t = f_T\left(f_G(\mathbf{X}^t)\right) \text{ or } f_T(\mathbf{X}^t)||f_G(\mathbf{X}^t) \quad (7)$$

Another strategy for sequentially encoding the hidden states incorporates $f_G(\cdot)$ into $f_T(\cdot)$ rather than stacking them. Since there are usually projection or convolution modules in $f_T(\cdot)$ to handle the features of nodes, this ''integrated

TABLE 3. Main components of selected (DT)DGNNs which sequentially encode hidden states. Without specification, GNN refers to graph convolution or message passing, OP refers to Orthogonal Procrustes [222], TCN refers to temporal convolution and its variants, Attention refers to the attention mechanism [24], AE refers Autoencoder and its variants, and RW refers to Random Walk-based approaches [132], [133], [143].

Model	Task case	Structure	Graph Encoding	Temporal Encoding	Embedded Object	Ref.
DyReg	Trans-fix$_{V,E}$	stacked	GNN	GRU	Node	[107]
RSTG	Trans-fix$_{V,E}$	stacked	GNN	LSTM	Node	[106]
GNN-RNN	Trans-fix$_{V,E}$	stacked	GNN	LSTM	Node	[68]
Graph WaveNet	Trans-fix$_{V,E}$	stacked	GNN	TCN	Node	[51]
STGCN	Trans-fix$_{V,E}$	stacked	GNN	TCN	Node	[60]
DCRNN, GCRNN	Trans-fix$_{V,E}$	integrated	GNN	GRU	Node	[61]
USSTN	Trans-fix$_{V,E}$	stacked	GNN	MLP	Node	[65]
GRNN	Trans-fix$_{V,E}$	stacked	GNN	Gated RNN	Node	[62]
GCRN-M1	Trans-fix$_{V,E}$	stacked	GNN	LSTM	Node	[42]
GCRN-M2	Trans-fix$_{V,E}$	integrated	GNN	LSTM	Node	[42]
ASTGCN	Trans-fix$_{V,E}$	stacked	GNN	TCN	Node	[102]
DetectorNet	Trans-fix$_{V,E}$	stacked	Transformer	1D Conv	Node	[136]
VDGCNeT	Trans-fix$_{V,E}$	stacked	Transformer	Transformer	Node	[101]
STAGIN	Trans-fix$_{V,E}$	stacked	GIN	Transformer	Graph	[108]
GraphSleepNet	Trans-fix$_{V,E}$	stacked	GNN	TCN	Graph	[69]
WD-GCN, CD-GCN	Trans-fix$_{V}$	stacked	GNN	LSTM	Node	[90]
tNodeEmbed	Trans-fix$_{V}$	stacked	RW	OP & LSTM	Node	[4]
DynSEM	Trans-fix$_{V}$	stacked	RW	OP	Node	[77]
GC-LSTM	Trans-fix$_{V}$	integrated	GNN	LSTM	Node	[105]
LRGCN	Trans-fix$_{V}$	stacked	GNN	LSTM	Path (set of edges)	[10]
E-LSTM-D	Trans-fix$_{V}$	stacked	AE	LSTM	Node	[110]
STAR	Trans-fix$_{V}$	stacked	GNN	GRU	Node	[103]
HTGNN	Trans-fix$_{V}$	stacked	Attention	Attention	Node	[104]
TNDCN	Trans-vary	stacked	GNN	TCN	Node	[88]
ANOMULY	Trans-vary	stacked	GNN	GRU	Node	[72]
DyHAN	Trans-vary	stacked	GNN	Attention	Node	[109]
THINK	Trans-vary	stacked	GNN	TCN	Node	[223]
CoEvoGNN	Trans-vary	stacked	GNN	Normalized sum	Node	[203]
DspGNN	Trans-vary	stacked	GNN	LSTM	Node	[204]
Euler	Trans-vary	stacked	GNN	LSTM	Node	[89]
ContrastEgo	Trans-vary	stacked	GNN	Transformer	Graph	[121]
Dyn-GCN	Ind$_{DG}$	stacked	GNN	Attention	Graph	[78]
DDGCN	Ind$_{DG}$	stacked	GNN	Temporal Fusion	Graph	[79]
DGTR	Ind$_{DG}$	stacked	Transformer	Transformer	Graph	[123]

mode'' turns these modules into $f_G(\cdot)$ to aggregate neighboring features. **GCRN-M2** [42] replaces the convolution in convLSTM with graph convolution. Similar examples are **GC-LSTM** [105] and **DCRNN** [61] which replace the linear layer in LSTM and GRU with respectively GCN [111] and diffusion convolution [112].

To deal with the addition/deletion of nodes in a transductive nature, i.e. *Trans$_{vary}$* case, one needs to handle V_t which may change across snapshots. **TNDCN** [88] proposes to set a universal node set $V = \cup V_t$ to ensure that $|V|$ is the same for each snapshot, which transforms the transductive case into a node set-invariant case on DTDG.

In the inductive case, one cannot presume V to set the universal node set, which brings about an inconsistent number of nodes in each snapshot making $f_T(\cdot)$ impossible to encode at the node level. Therefore, this method in inductive tasks is only applicable for encoding graph-level representations across time, e.g. for fake news detection based on its propagation tree. **Dyn-GCN** [78] applies Bi-GCN [113] to encode the hidden states $\mathbf{z}_{G_t}$ for each snapshot t by aggregating the hidden states of its nodes and edges, and passes $(\mathbf{z}_{G_1}, \mathbf{z}_{G_2}, \ldots, \mathbf{z}_{G_T})$ into an attention layer to compute the final hidden states of the entire DG.

D. SEQUENTIALLY ENCODING PARAMETERS THETA

Although *enc*($\mathbf{H}$) is relatively intuitive and simple in terms of model structure, the problem is that they can neither handle the frequent changes of the node set, especially in the inductive tasks, nor pass learned parameters of $f_G(\cdot)$ across time steps [3]. To encode DTDGs more flexibly, some other approaches constrain or encode the parameters of $f_G(\cdot)$ across time steps [3], [114], [115].

In order to encode the parameters Θ of GCN, **EvolveGCN** [3] proposes to use LSTM or GRU to update the parameters of the GCN model at each time step as shown in (8) and (9):

$$\Theta_{f_G}^t = \text{LSTM}(\Theta_{f_G}^{t-1}) \tag{8}$$

$$\Theta_{f_G}^t = \text{GRU}(\mathbf{H}^t, \Theta_{f_G}^{t-1}) \tag{9}$$

Otherwise, by constraining the GNN parameters, **Dyn-GEM** [114] incorporates autoencoder (AE) to encode and reconstruct the adjacency matrix of each snapshot. Its parameters Θ_t are initialized with Θ_{t-1} to accelerate and stabilize the model training. Similarly, in **VGRNN** [115] the authors combine GRNN and a variational graph AE in order to reuse learned hidden states of t^- to compute the prior distribution parameters of the AE.

E. TIME EMBEDDING

When the scale of the dynamic graph is large, as in social networks and recommendation systems, aggregation to snapshots is neither precise nor efficient [45], [50]. Such DGs are thus represented by a set of timestamped events. Therefore, when encoding a CTDG, one should consider not

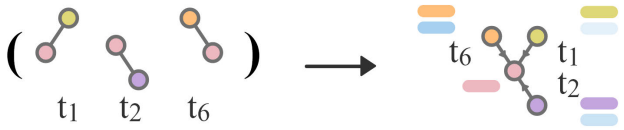


FIGURE 9. To aggregate information of neighbors on a dynamic graph, for example, to encode the pink node, the nodes with which it has an edge, i.e. its *temporal neighbors* [93] (yellow, orange, purple nodes) and the temporal information (in blue) are embedded as vectors. This can also be interpreted as constructing an *equivalent static (sub)graph* via attributes.

only how to asynchronously update the node representations over time, but also to define the neighbors of nodes.

The DGNNs presented in this subsection update the representation of a node v when it changes, i.e., when it participates in a new edge or when its attributes change. Its neighbors at moment t, also called *temporal neighborhood*, are usually defined as the nodes that have common edges with v (before t) [6]. Thus, such models consider v-centric equivalent static subgraph as shown in Fig. 9. In such cases, the occurrence time of each edge can be considered as part of the edge attributes or used as the weight for graph convolution.

TDGNN [83] proposes a weighted graph convolution by assuming that the earlier an edge is created, the more weight this edge will have in aggregation, as shown in Equ. 10. In more detail, the weight $\alpha^t_{u,v}$ of an edge (u, v) at moment t is calculated by the softmax of the existence time $t - t_{u,v}$ of the edge.

$$\alpha^t_{u,v} = \frac{e^{t-t_{u,v}}}{\sum_{u\in N_t(v)\cup v} e^{t-t_{u,v}}} \tag{10}$$

To better leverage the time information, **DySAT** for DTDGs [116] directly learns position embedding [117] to transform the absolute temporal position of snapshot t into a vector of d_v dimensions and add it to the node's hidden state $\mathbf{h}_v \in \mathbb{R}^{d_v}$. **TGAT**, on the other hand, uses function time encoding [118] to ensure the translation-invariant of the time kernel function. It embeds the value of the existence time $t - t_{(u,v)}$ of edge (u, v) on the CTDG as a vector of d_T dimensions and then concatenates it to the node v's hidden states $\mathbf{z}^t_v$. When aggregating neighbor information for a node u, the weight of each neighbor is computed by multi-head attention (11). Similar methods have also been applied for time embedding in Transformers on Dynamic Graphs [73], [98], [99], [100], [101], [116], [119], [120], [121], [122], [123], [124], [125], [126], [127]. Section III-G delves into a detailed exploration of position and time embedding approaches in transformer architectures.

$$\begin{aligned} rCl\mathbf{q}(\mathrm{t}) &= [\mathbf{Z}(\mathrm{t})]_0\mathbf{W}_\mathrm{Q} \\ \mathbf{K}(\mathrm{t}) &= [\mathbf{Z}(\mathrm{t})]_{1:\mathrm{N}}\mathbf{W}_\mathrm{K} \\ \alpha^t_{u,v} &= \frac{\exp\left(\mathbf{q}^T_u\mathbf{k}_v\right)}{\sum_{v\in N_t(u)}\exp\left(\mathbf{q}^T_u\mathbf{k}_v\right)} \end{aligned} \tag{11}$$

To better reuse the messages aggregated for each node by TGAT, **TGN** [2] adds a memory mechanism that memorizes the historical information for each node v with a memory vector $\mathbf{s}_v$. This memory is updated after each time t a node v aggregates its neighbors' information $\mathbf{m}^t_v$.

$$\mathbf{s}^t_v = \mathrm{MLP}(\mathbf{m}^t_v, \mathbf{s}^{t^-}_v) \tag{12}$$

Similar to TGN, **TGNF** [8] embeds nodes and time information via TGAT [6], and then updates the node's memory $\mathbf{S}$ via the temporal memory module (TMM). In order to learn variational information better, it calculates the similarity $(\mathbf{S}_t, \mathbf{S}_{t^-})$ of the memory $\mathbf{S}$ at t and t^- during training as part of the loss, called Time Difference Network (TDN).

Besides the memory module, later studies attempted to enhance the performance of TGAT from various perspectives. For instance, to accelerate the speed of model inference of TGAT, especially in contexts like financial fraud detection, **APAN** [128] redesigned the TGAT's workflow. They transitioned from the conventional encoding→propagation→decoding architecture of TGAT to an encode→decode→propagation sequence, thus effectively decoupling model inference from graph computation. On the other hand, in the pursuit of architectural simplicity, **Graph Mixer** [129] posits that a static functional time encoding is sufficient, negating the need for additional training. Further emphasizing simplicity, this model relies on MLP and mean-pooling layers, avoiding the complexity introduced by structures such as RNNs or multi-head attention mechanisms.

Some other approaches consider the interaction predictions on the graph as a Temporal Point Problem (TPP [130]) and simulate the conditional intensity function that describes the probability of the event. **DyRep** [43] encodes a strength matrix $\mathbf{S} \in \mathbb{R}^{|V|\times|V|}$ to simulate the intensity function of the interactions between each node pair and uses $\mathbf{S}$ as the weights for graph convolution. $\mathbf{S}$ is initialized by the adjacency matrix $\mathbf{A}$ and updated when an interaction (u, v) occurs at time t. In this case, the embedding of each node involved is updated. For example, the update of v is the sum of the three embedding components given by (13): the latest embedding of v, the aggregation of embeddings of u's neighbor nodes, and the time gap between v's last update and this update.

$$\mathbf{z}^t_v = \sigma\left(\mathbf{W}_1\mathbf{h}^{t^-}_{\mathrm{N(u)}} + \mathbf{W}_2\mathbf{z}^{t^-}_v + \mathbf{W}_3(t - t^-)\right) \tag{13}$$

All of the models mentioned above embed timestamps or time gaps as part of the features. However, models such as TGAT are unable to capture accurate changes in structure without node features [17]. As discussed above, DyREP [43] solves this problem but is unable to perform the inductive task as it relies on the intensity matrix $\mathbf{S}$. This raises another challenge: How to learn in an inductive task based only on the topology when the nodes have no features.

F. CAUSAL RANDOM WALKS

Random walk-based approaches do not aggregate the neighborhood information of nodes, but sample node sequences to

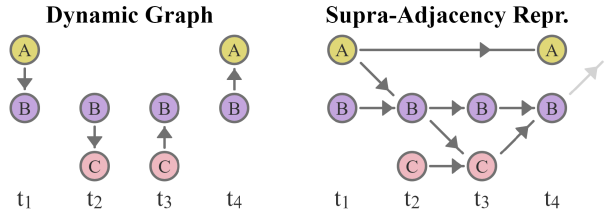


FIGURE 10. Supra-adjacency representation: treating discrete time dynamic graphs as directed static graphs for sampling random walks.

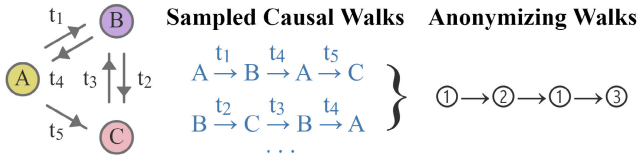


FIGURE 11. Schematic diagram on sampling casual walks on a CTDG, the right part shows graph pattern extracted by anonymizing nodes.

capture the local structure. To incorporate time information into the random walks, different ways of defining "causal walks" on dynamic graphs are derived.

To sample random walks with temporal information in discrete time, an intuitive way is to convert the DTDG into a directed static graph. Huang et al. [131] construct a *ESG via topology*, and allow random walks to sample inside the current layer (/timestep) to get the structural information or walk into the previous layer to obtain historical evolving information. Following the supra-adjacency representation (see fig. 10), **DyAne** [18] applies the random walk approach of static graphs (DeepWalk [132] and LINE [133]) to DTDGs. Otherwise, without using these representations, **LSTM-node2vec** [94] directly samples one neighboring node of a target node v per snapshot as a walk.

Similar to the method based on supra-adjacency, the walks obey causality under continuous time, i.e. after walking from node B to node C via an edge occurring at t_2, node C can only walk to the next node via an edge having occurring time $t > t_2$ (Fig. 11).

Some early methods are **CTDNE** [84] and **T-Edge** [95]. To encode a node v, they both sample the causal walks starting from v, each walk W being a sequence of (node, time) pairs. Then they embed each (node, time) pair and encode the sequence W by $f_T(\cdot)$ like RNNs. Combined with anonymous walk embedding, Causal Anonymous Walks (**CAW** [17]) anonymize nodes and their attributes in order to focus more on graph motifs, which solve the problem at the end of Sec. III-E. Drawing inspiration from CAW, **NeurTWs** [134] proposes spatio-temporally biased random walks that refine the walk sampling algorithm. This adjustment elevates the traversal probability of the most recent neighbors and those with superior connectivity. Similarly anchored in CAW, **CAt-Walk** [135] extends random walk sampling to hyper-graphs. This is achieved by sampling set walks to encode hyperedges, where notably, an edge can bridge multiple nodes and is depicted as a set.

G. DYNAMIC GRAPH TRANSFORMERS

As a model originally designed for sequence to sequence problems, Transformers architectures [23] are increasingly being employed in static graph representation learning and extended to dynamic graphs. We have already mentioned above how models use Transformer as a temporal [101], [108], [121], [123] or spatial encoder [101], [123], [136], as well as implementing the positional encoding technique, all of which show its great potential for learning dynamic graph representations.

Existing surveys [137], [138] have discussed the integration of Transformers with Graph Neural Networks (GNNs) for static graphs. This integration is accomplished in three principal ways [138]: (a) Using GNNs as auxiliary modules, (b) Improving Positional Embedding (PE) with graphs, and (c) Improving Attention matrix with graphs. However, on dynamic graphs, both PE and attention mechanisms become far more complex with the addition of the time dimension. To our knowledge, there is no current survey that compares these model architectures. In this subsection, we review cutting-edge dynamic graph transformers and synthesize their techniques in table 4.

1) COMBINATION OF TRANSFORMER WITH DGNNS

There are three possible ways for combining Transformers and DGNNs models, synthesized in Fig. 12 (left). The first combination is a sequential application of transformer layer(s) and GNN layer(s) in an independent way [101], [121], [124], [139]. The second combination alternates GNN layers and Transformer layers in the DG encoder [100], [126], [136]. The third combination is a parallel encoding of the DG by independent transformer and GNN layers, followed by a combination of their encoded hidden states [120], merging the strengths of both layers. Additionally, some dynamic graph models [73], [98], [99], [122], [123], [125], [127] use transformers exclusively as graph encoders, exploiting the self-attention mechanism for node hidden states propagation without relying on traditional GNN architectures.

2) POSITIONAL ENCODING (PE) ON DYNAMIC GRAPHS

Positional Encoding (PE) is a central and essential question in the implementation of transformers. In the context of dynamic graphs, the PE is far from trivial because it must account for the position of the elements in the graph structure, as well as their temporal location. We have distinguished position encoding into 4 types. The first method is the embedding based on the time-related index of nodes or edges [98], [100], [120], [124], [127]. The second method utilizes (global) graph spectral information [99], such as the Laplacian or its eigenvectors. The third method relies on spatial domain information or local structures, such as embedding the node degrees or random walks [73], [99], [101], [122], [123], [127], [139]. Finally, as mentioned in

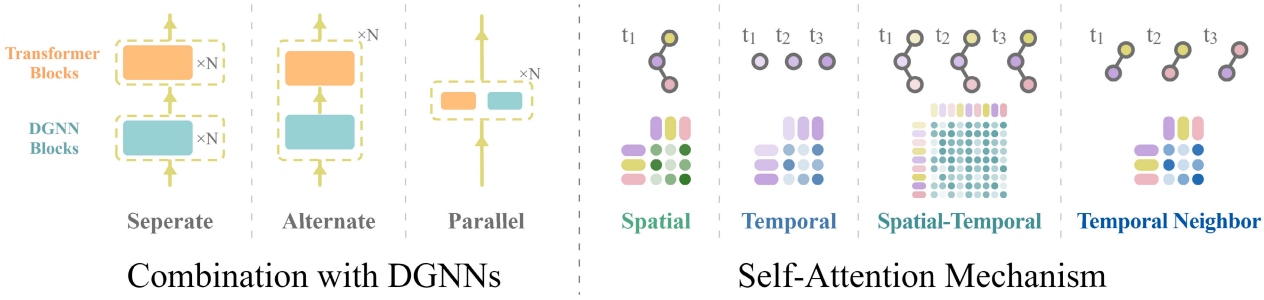


FIGURE 12. Left: Three ways of combining Transformers with DGNNs. Right: The integration of Self-Attention (SA) for spatial attention, temporal attention, spatial-temporal attention, or temporal neighbor attention.

TABLE 4. An analysis of the architecture of Dynamic Graph Transformers: The Transformer architecture can be combined with DGNN layers separately, alternately, in parallel, or even completely replace the DGNN module. Positional Encoding can be performed on the chronological index, on spectral and spatial features, or on the timestamp. The Self-Attention (SA) mechanism can operate spatially as a graph encoder (S), or temporally as a temporal encoder (T). Otherwise, SA can also be performed on the replicas of nodes across multiple snapshots (ST), called Spatial-Temporal Self-Attention, or on the temporal neighbors of different timestamps (Temporal NB).

Model name	Ref.	Input Type	Combination with DGNNs			Positional Encoding				Self-Attention Mechanism			
			Separate	Alternate	Parallel	Index	Spectral	Spatial	Time	S	T	ST	Temporal NB
Spacetimeformer	[98]	Signals				✓			✓			✓	
TENET	[120]	Signals			✓	✓			✓	✓	✓		
DetectorNet	[136]	STGs		✓						✓	✓		
ASTTN	[99]	STGs					✓	✓	✓			✓	
STSGT	[100]	STGs		✓		✓			✓			✓	
VDGCNeT	[101]	STGs	✓					✓	✓	✓	✓	✓	
TADDY	[73]	DTDGs						✓	✓				✓
ContrastEgo	[121]	DTDGs	✓						✓	✓			
DGT	[122]	DTDGs						✓	✓	✓			
DGTR	[123]	DTDGs						✓	✓	✓	✓		
RDGT	[139]	DTDGs	✓					✓				✓	
TGT	[124]	CTDGs	✓			✓			✓				✓
HTGT	[125]	CTDGs							✓	✓	✓		
Todyformer	[126]	CTDGs		✓					✓				✓
TCL	[127]	CTDGs				✓		✓	✓				✓

section III-E, almost all the dynamic graph transformers use time embedding to represent the timestamp or snapshot index with a vector.

3) SELF-ATTENTION (SA) MECHANISM

Concerning Self-Attention (SA), there are four solutions on dynamic graphs, as shown in Fig. 12 (right): **Spatial SA**: when a token represents a node in a snapshot, SA operates at the spatial level, equivalent to the graph encoder f_G, with SA matrix dimensions being the square of the number of nodes N_t in the snapshot t [101], [120], [121], [122], [123], [125], [136]. **Temporal SA**: When a token represents the same node (or the entire snapshot) across multiple snapshots, SA operates at the temporal level and acts as a temporal encoder f_T, its SA matrix dimensions thus are the square of the number of time steps K [101], [120], [123], [125], [136]. **Spatial-Temporal SA**: Following Temporal Edge Modeling III-B, creating replicas for each node in K snapshots, forming Spatial-Temporal Attention, with the Attention matrix size being the square of $\sum N_t$ or $K \times T$ [98], [99], [100], [101], [139]. **Temporal Neighbor SA** also operates on topology, but its tokens are nodes/neighbors from different timestamps [73], [124], [126], [127], i.e. temporal neighbors in CTDGs [2], [6]. Timestamp information is combined into hidden states by time embedding, therefore no replica of nodes is needed to be created for each node as in Spatial-Temporal SA.

The emergence of Transformer architectures tailored to dynamic graphs marks an evolution towards more diversified temporal encoding methods. Such as in **ASTTN** [99] which integrates multiple types of positional encoding and learns spatiotemporal attention, as well as in **VDGCNeT** [101] which combines different attention mechanisms. Meanwhile, the effectiveness of each positional encoding approach and attention mechanisms in different cases and datasets poses an open challenge that warrants further analysis and research.

H. ENCODING HETEROGENEOUS GRAPHS

Since heterogeneous graphs can maintain more informative representations in tasks like link prediction in recommendation systems, they present a unique challenge in terms of node embedding on dynamic graphs. A key concern is to have a dedicated module for handling different types of nodes and edges on dynamic graphs. We introduce in the following paragraphs two main approaches as illustrated in Fig. 13.

GNN-based models keep the same dimension d of embedding for different node types (e.g. for user nodes and item nodes) to facilitate computations. For example,

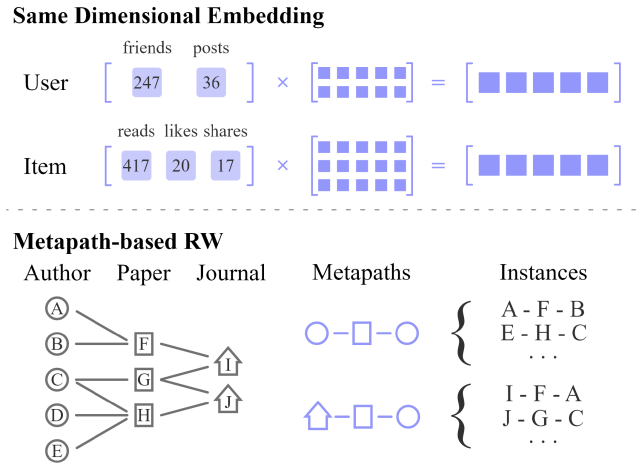


FIGURE 13. Handling heterogeneous nodes. Above: embedding them into the same vector space (e.g. with $d = 5$). Below: sampling random walks with defined metapaths.

to update the node embeddings when user u and item i have a connection $e^t_{u,i}$ at time t, **JODIE** [82] embeds the user attributes $\mathbf{x}_u$, the item attributes $\mathbf{x}_i$, the edge attributes $\mathbf{x}_e$ and the time difference since the last update Δ to the same dimension d and then adds them together:

$$\begin{aligned}\mathbf{h}^t_u &= \sigma\left(\mathbf{W}^u_1\mathbf{h}^{t^-}_u + \mathbf{W}^u_2\mathbf{h}^{t^-}_i + \mathbf{W}^u_3\mathbf{h}^t_e + \mathbf{W}^u_4\Delta_u\right)\\ \mathbf{h}^t_i &= \sigma\left(\mathbf{W}^i_1\mathbf{h}^{t^-}_i + \mathbf{W}^i_2\mathbf{h}^{t^-}_u + \mathbf{W}^i_3\mathbf{h}^t_e + \mathbf{W}^i_4\Delta_i\right)\end{aligned} \tag{14}$$

where t^- refers to the timestamp of the last update thus $\Delta = t - t^-$, and $\mathbf{h}^t_i = \mathbf{x}^t_i$ for the first layer of model. **DMGCF** [80] and **DGCF** [81] also use similar approaches for same dimensional embedding. In line with this method, DyHAN [109] and HTGNN [104] improve this approach by assigning different weights to neighbors of different types during the aggregation process, both methods first use a graph convolution for each type of neighbors surrounding node v. Then, an attention mechanism is employed to compute the weight of each type for updating the hidden states of v.

RW-based methods deal with heterogeneous graphs by defining metapaths [140] which specifies the type of each node in the walk, such as (user, item, user), so that each random walk sampled (also called "instance") with the same metapath can be projected to the same vector space. Examples on the dynamic graph are **THINE** [141] and **HDGNN** [142] which sample instances and encode them through the attention layer and bidirectional RNN layers, respectively.

So far, we have presented the main approaches for capturing time and graph dependencies within dynamic graphs, with a special emphasis on the cutting-edge area of Dynamic Graph Transformers. In the next section, we propose guidelines for the design and optimization of DGNN architectures, as well as present several tables of common public datasets for DGNN evaluation.

TABLE 5. Adapted cases of dynamic graph predictive tasks of each method: **TE** stands for "Temporal Edge Modeling", **Enc(H)** for "Sequentially Encoding Hidden States", **Enc(Θ)** for "Sequentially Encoding Model Parameters", **Emb(t)** for "Embedding time", **Causal RW** for "Causal Random Walks", **DGT** for "Dynamic Graph Transformer". ✓ means "applicable", ∗ means "applicable with output restrictions", and no symbol indicates that it is not yet used in the literature.

Approach	DT Trans.	DT Ind.	CT Trans.&Ind.
① TE	✓		
② Enc(H)	✓	∗	
③ Enc(Θ)	✓	✓	
④ Emb(t)	✓	✓	✓
⑤ Causal RW	✓	✓	✓
⑥ DGT	✓	✓	✓

IV. DISCUSSION

In the previous sections, we have highlighted the diversity of contexts that can be encountered when considering machine learning on dynamic graphs, as well as the diversity of existing models to tackle these problems. In this section, we present some guidelines for designing DGNNs based on the taxonomies presented in section II and III. To the best of our knowledge, while such guidelines have already been proposed for static graphs [31], they do not exist for DGNNs.

A. HOW TO DESIGN DGNNS?

For static graphs, Zhou et *al.* [31] described the GNN design pipeline as: *i)* determine the input graph structure and scale, *ii)* determine the output representation according to the downstream task, and *iii)* add computational modules.

For dynamic graphs, the design of DGNN has to consider more factors. We therefore generalize the workflow of designing DGNN as follows:

1) Clearly define the input, output and nature of the task, according to the taxonomies of section II;
2) Choose the compatible time encoding approach according to the learning setting, using the categorizations of section III and more precisely the Table 5;
3) Design NN structure;
4) Optimise the DGNN model.

The key points in the DGNN compatibility are the input time granularity, the nature, and the object to be encoded. We concluded their known adapted DG types and listed them in table 5 and describe the various cases in this subsection.

Transductive tasks under discrete time, as a relatively simple setting, can be encoded with any approach to incorporate time information. In the setting of inductive DT, no approach using *TE* is found in the literature, and the method of *enc*(**H**) must also have its output aggregated because of the gap mentioned in sections III-C, e.g. node-level time-aggregated or graph-level time-step. To handle continuous time, only *emb(t)* and *Causal RW* are widely used in the literature.

Once the approach has been selected, the next step is to add the computational components, which are very different for each approach.

For the first three methods mainly used in discrete time, i.e., ① *TE*, ② *Enc*(**H**), and ③ *Enc*(Θ), they can be abstracted to encode graph and time information via $f_G(\cdot)$ and $f_T(\cdot)$, respectively. These neural network components are usually some of the modules introduced in the appendices A, B and Tab. 3. In particular, the encoding of temporal edges in the *TE* approach can be performed by $f_G(\cdot)$ without a specific $f_T(\cdot)$. In *enc*(**H**) approach, there are more possible ways to combine $f_G(\cdot)$ and $f_T(\cdot)$ (stacked, integrated, etc.). In *enc*(Θ) approach, the key is how $f_T(\cdot)$ acts on the parameters of $f_G(\cdot)$. Remarkably, compared to *Enc*(**H**) or *Enc*(Θ) which encode a graph of $|V_t|$ nodes T times, the *TE* method encodes a large graph with $\sum_{t=1}^{T}|V_t|$ nodes at once. Its high space complexity is a possible problem.

For the time embedding approach ④ *emb(t)*, in addition to the method of aggregating information of neighbors i.e. $f_G(\cdot)$, one also needs to consider the function for time embedding (e.g. by a set of sine or cosine functions [6] or a learnable linear layer [43]) and how to combine them to the hidden states of edges or nodes (e.g. by addition or concatenation). In the situation where the past representation of a node is stored [2], [8], [43], one also needs to determine the module used to update the node representation, i.e. $f_T(\cdot)$.

For random walk-based method ⑤ *Causal RW*, one needs to determine the random walk sampling strategy as well as the method for encoding the sampled walks, typical walk-encoding strategies are [132], [133], [143], and [144]. Methods based on *emb*(*t*) and *CausalRW* are increasingly being applied to CTDGs [2], [17], [118] because there are no limitations due to the case of DG predictive task.

⑥ Dynamic Graph Transformers are also capable of encoding dynamic graphs in various cases, but it's important to consider their integration with other DGNN modules, as well as the approach for the position encoding and the Self-Attention mechanism, as described in Section III-G. From the point of view of computational efficiency, the quadratic complexity due to its Self-attention mechanism is also a problem to be considered.

Last but not least, in the case of heterogeneous graphs, an additional vector projection or setting of metapaths has to be considered.

B. HOW TO OPTIMIZE DGNNS?

With the development of artificial neural networks and the continuous emergence of new structures over the last five years, questions have been raised about the optimization of DGNNs. Apart from general neural network training issues (overfitting, lack of data, vanishing gradient, etc.), a main issue for GNN is over-smoothing [145], [146], [147]: if the number of layers and iterations of a GNN is too large, the hidden states of each node will converge to the same value. The second main challenge is over-squashing [148], [149], [150]: if a node has a very large number of K-hop neighbors, then the information passed from a distant node will be compressed and distorted.

To overcome the above problems, numerous methods have been proposed. According to us, general trends to improve DGNN can be categorized as (1) Input oriented, (2) DGNN component oriented, and (3) DGNN structure oriented.

1) INPUT ORIENTED OPTIMISATION

To avoid overfitting when learning graph representations, two main issues related to the DG need to be considered: the noise in topology (e.g. missing or incorrect links) and the noise in the attributes (e.g. incorrect input attributes or output labels) [151], [152].

Since there may be pairs of similar nodes in the graph that should be connected but are not (due to geographical constraints, missing data, etc.), some approaches aim to exploit a more informative graph structure or to augment attribute data.

For discrete time, an example is **ST-SHN** [67] for crime prediction. Considering each region as a node and its geographical connectivity as an edge, ST-SHN infers the hyper-edges connecting multiple regions by learning the similarity of hidden states between node pairs. These hyper-edges help to learn cross-region relations to handle the global context.

For continuous time, such as heterogeneous graph-based recommendation systems, **DMGCF** [80] constructs two additional homogeneous graphs G_u for users and G_i for items based on the known user-item graph G_{ui}. Then two GCNs are used to aggregate information on $G_u \cup G_i$ and G_{ui} respectively to learn more informative node embeddings.

Noise in attributes can also lead to overfitting DGNNs, therefore adaptive data augmentation is another direction for input-oriented improvement. Wang et al. proposed Memory Tower Augmentation (**MeTA** [153]) for continuous time data augmentation by perturbing time, removing edges, and adding edges with perturbed time. Each augmentation has learnable parameters to better adapt to different input data.

2) DGNN COMPONENT ORIENTED OPTIMISATION

To solve the over-smoothing and over-squashing problems on graphs, the improvement of the DGNN modules focuses on a more versatile message propagation and a more efficient aggregation.

To avoid stacking multiple layers of GCNs, **TNDCN** [88] uses different-step network diffusion which provides a larger receptive field for each layer. Each step k propagates attributes with a k-hop neighborhood and has its independent learnable parameters Γ_k, as shown in equ. 15, where $\widetilde{\mathbf{A}}^{\mathbf{k}}$ refers to the parameterised k-hop connectivity matrix.

$$\mathbf{H} = \sum_{k\geq 0} \widetilde{\mathbf{A}}^{\mathbf{k}} \mathbf{H}^{k} \Gamma_k \tag{15}$$

Inspired by the Bidirectional LSTM [154], an easy-to-implement enhancement for propagation is bi-directional message passing **Bi-GCN** [78], [113]. It processes an undirected tree graph as two directed tree graphs: the first one

TABLE 6. Public datasets of node/edge-level predictive tasks. The ⋄ markers indicate that some of the publications use a subset of this data.

Source	Graph type	Data set	#Nodes	#Links	Duration	Publications
SNAP [168]	Social Network	Reddit	11,985	672,447	1 month	[2], [3], [6], [81], [82] [17], [122], [127]
	Social Network	Twitter	8,861	119,872	7 days	[2]
	Social Network	CollegeMsg [211]	1,899	59,835	6.5 months	[4], [127], [139]
	User-Item	Wikipedia	9,227	157,474	1 month	[2], [6], [81], [82] [17], [122], [127]
	User-Item	MOOC	7,145	441,749	1 month	[17], [126]
	User-Item	LastFM	2,000	1,293,103	1 month	[81], [82], [126], [127]
	Transaction Network	Bitcoin-Alpha [206], [207]	3,783	24,186	5 years	[3], [73], [84], [203], [204]
	Transaction Network	Bitcoin-OTC [206], [207]	5,881	35,592	5 years	[3], [83], [203], [204]
	Internet Interaction	Autonomous Systems [208]	6,474	13,233	1 month	[3], [73], [114]
	Collaboration Network	HEP-TH ⋄ [212]	27,770	352,807	10 years	[4], [114], [115], [139]
	Citation Network	DBLP ⋄ [213]	317,080	1,049,866	unknown	[80], [90], [141]
	Face-to-face interaction	Resistance (62 graphs) [209], [210]	451	3,126,993	40 minutes	[110]
Network Repository [169]	Email Network	Radoslaw	167	82.9K	9 months	[83], [84], [110], [122]
	Email Network	Enron ⋄ [170]	69.2K	274.6K	unknown	[84], [110], [114]–[116] [17], [83], [122], [126]
	Email Network	Email-EU [169]	986	332,334	2.5 years	[84]
	Face-to-face interaction	CONTACT	274	28,2K	4 days	[84], [110]
	Face-to-face interaction	HYPERTEXT09	113	20,8K	3 days	[83], [84]
	Social Network	FB-Forum ⋄	899	33.7K	6 months	[83], [84], [110]
	Social Network	Wiki-Elec	7.1K	107K	3.5 years	[83], [84]
	Message Network	Email-DNC [169]	1,866	39,264	1 year	[73]
KONECT [214]	Social Network	UC Irvine forum [224]	1,899	59,835	6 months	[3], [17], [73], [116] [122], [126]
	Social Network	Facebook wall posts	46,952	876,993	4.5 years	[4]
	Social Network	Facebook friendships	63,731	817,035	2 years	[4]
	Social Network	Slashdot	51,083	140,778	unknown	[4]
	Social Network	Digg [221]	30,360	87,628	15 days	[73]
	Citation Network	Colab [217]	315	5,104	unknown	[89], [115]
	Transaction Network	Elliptic [218]	203,769	234,355	2 years	[3]
MIT [a]	Physical Proximity	Social Evolution [225]	83	2,016,339	1 year	[17], [43], [115], [126]
Yelp [b]	User-Item	Yelp ⋄	>1.9 M	>7M	Since 2004	[80], [116], [122], [141], [219]
Grouplens [c]	User-Item	MovieLens ⋄	224K	25M	Since 1997	[80] [116], [122], [204]
Facebook	Social Network	FB [220]	663	23,394	unkown	[89], [115]

[a] realitycommons.media.mit.edu
[b] https://www.yelp.com/dataset
[c] https://grouplens.org/datasets/movielens/

from the root to the leaves, and the second one from the leaves to the root. Then two GCNs with different parameters encode each case independently to obtain more informative hidden states.

Besides improving propagation methods, a widely used technique for aggregating information is the (multi-head) attention mechanism, which enables the model to use adaptive weights $\left(f_{attn} : \mathbb{R}^{|N_{v_i}|\times d} \rightarrow \mathbb{R}^{|N_{v_i}|\times 1}\right)$ for enhancing vanilla GCNs [155], as well as its variant self-attention mechanism $\left(f_{self-attn} : \mathbb{R}^{|N_{v_i}|\times d} \rightarrow \mathbb{R}^{|N_{v_i}|\times d'}\right)$ which is good at capturing the internal correlation among elements in the sequence.

The three component-oriented improvements mentioned above are applicable not only to $f_G(\cdot)$, but also to $f_T(\cdot)$ when encoding along time, such as dilated convolution for expanding the temporal perceptual field [157], Bi-LSTM for bidirectional propagation along the sequence [154], and (self-)attention mechanisms for encoding sequences [23].

3) DGNN STRUCTURE ORIENTED OPTIMISATION

Another research direction is to optimize the overall structure. For example, residual connection [158] for dealing with the vanishing gradient is widely used in DGNNs, especially for the DGNNs which sequentially encode hidden states [61], [65], [90].

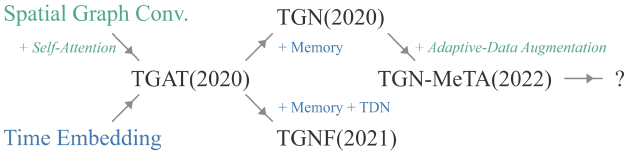


FIGURE 14. The main differences between the time-embedding methods, with the green color indicating the graph encoding-related methods and the blue color indicating the time encoding-related methods.

In particular for the time embedding approach under continuous time, how to better store historical information and how to learn variations of information are two major challenges due to the difficulty of encoding over time. Models such as **TGAT** [6] can encode the timestamps of the event but cannot reuse the hidden states already encoded in the past timesteps. **TGN** [2] addresses this problem by incorporating memory modules, **TGNF** [8] adds the similarity between current memory and previous memory in the loss to encourage the model to learn variational information. All these improvements enhance the performance and efficiency of the model. The evolution of the relevant models is shown in Fig. 14.

TABLE 7. Publicly available datasets for the graph-level predictive tasks.

Source	Graph type	Data set	#Graphs	#Average Nodes	Publications
Weibo [159]	Propagation Graph	Weibo	4657	804	[8], [78], [79], [113], [123], [216]
Twitter [160]	Propagation Graph	Twitter15	1490	222	[8], [78], [113], [216]
Twitter [161]	Propagation Graph	Twitter16	818	250	[8], [78], [113], [216]
Fake News Net [162]	Propagation Graph	Fake News Net	4168	42	[8], [123]
Pheme [163]	Propagation Graph	Pheme	5748	16	[79]
University of Florence	Skeleton Based Action	Florence 3D	215	15	[92]
University of Bonn	Skeleton Based Action	HDM05 [166]	2,337	31	[92]
Nanyang Technological University	Skeleton Based Action	NTU RGB+D [164], [165]	56,880	25	[92]
Humanconnectome	Biometric Signals	HCP ⋄ [108]	N/A	N/A	[108], [120]
Montreal Archive of Sleep Studie	Biometric Signals	MASS-SS3 [167]	N/A	N/A	[69]

TABLE 8. Publicly available datasets for geosignal predictive tasks.

Source	Graph type	Data set	#Nodes	#Edges	Granuality	Duration	Publications
[226]	Traffic Network	METR-LA	207	1515	5 minutes	4 months	[51], [61]–[63], [98], [99], [101], [136]
Caltrans [a]	Traffic Network	PeMS-BAY	325	2369	5 minutes	6 months	[51], [60], [61], [63], [98], [99], [101], [136]
NREL of U.S.	Energy Network	Solar-energy	137	N/A	10 minutes	52560 steps	[63], [215]
UCI	Energy Network	Electricity	321	N/A	hourly	26304 steps	[63]
New York Times[b]	Influenza Signals	NYT COVID-19	3143	N/A	daily	Since 2020	[65], [100]
niid.go.jp	Influenza Signals	IDWR in Japan	47	N/A	weekly	6.5 years	[64]
gis.cdc.gov	Influenza Signals	US-HHS ILINet	10	N/A	weekly	15 years	[64]
gis.cdc.gov	Influenza Signals	CDC in US	49	N/A	weekly	7 years	[64]

[a] https://dot.ca.gov/programs/traffic-operations/mpr/pems-source
[b] https://github.com/nytimes/covid-19-data

C. DATASETS FOR EVALUATING DGNNS

Evaluation of dynamic graph neural networks is important for validating their performance in real-world scenarios. Proper evaluation not only proves a model's effectiveness but also demonstrates its generalizability across multiple datasets.

DGNNs are adept at handling a variety of tasks. Key tasks include node/edge-level and graph-level predictive tasks, as well as link prediction which predicts the probability of a connection between two nodes. The tasks and corresponding evaluation metrics are discussed earlier in sections II-D and II-F. A synthesis of the task modeling (classification or regression, discrete-time or continuous-time, transductive or inductive), metrics, and concrete applications (e.g., rumor detection, traffic flow forecasting) can be found in Table 2.

Given the complex form of dynamic graphs, to our knowledge, there is little mention of dataset compilation in existing surveys [45], [46], [47], only [44] provides a list of public datasets up to 2020. In order to better evaluate the performance of DGNNs, a selection of representative datasets from different domains is provided in three detailed tables, each highlighting a different area of focus: Table 6 describes the commonly used publicly available datasets for node/edge-level predictive tasks, the domains in which they were collected, and a selection of articles using the datasets. Table 7 focuses on graph-level tasks, including graph propagation for rumor detection [159], [160], [161], [162], [163], skeleton-based action recognition [164], [165], [166], and action or sleep stage classification based on biometric signals such as electrocardiogram (ECG) or electroencephalogram (EEG) [108], [167]. Table 8 focuses on the prediction of signals on geographically based graphs. Since the raw data is based on signals, in some literature [63], [64], [68], the number of edges is not determined by the dataset, but is calculated based on the geographic distance. We also provide the temporal granularity of the signal and the duration for each dataset.

In addition to the *SNAP* [168] and *Network Repository* [169] mentioned in the table 6, the latest libraries *TGB* [171] and *DyGLib* [172] have also begun to focus on the issue of dynamic graph benchmarking and datasets. These libraries aim to develop a solid pipeline to ensure the fairness and reproducibility of experiments performed on continuous-time dynamic graphs. The constant emergence of new libraries and open datasets symbolizes the potential of dynamic graph neural networks in various fields.

V. CONCLUSION

Thanks to their ability to integrate both structural and temporal aspects in a compact formalism, dynamic graphs have emerged in the last few years as a state of the art model for describing dynamic systems.

Many scientific communities have investigated this area of research using their own definition, their own vocabulary, their own constraints, and have proposed prediction models dedicated to their downstream tasks. Among these models, Dynamic Graph Neural Networks occupy an important place, taking benefit of the representation learning paradigm.

The first part of this survey provides clarification and categorization of the different dynamic graph learning contexts that are encountered in the literature, from the input point of view, the output point of view, and the learning setting (inductive/transductive nature of the task). This categorization leads to five different learning cases covering the different contexts encountered in the literature.

Using this categorization, our second contribution was to propose a taxonomy of existing DGNN models.

We distinguish six families of DGNN, according to the strategy used to incorporate time information in the model

Finally, we also provide practitioners with some guidelines for designing and improving DGNNs, as well as a summary of common public datasets across domains for validating their models.

As potential future research directions, we identify several open challenges beyond the commonly addressed issues of expressiveness and practical applications in surveys [44], [47].

1) The **evaluation of Discrete Time Dynamic Graphs** (DTDGs) warrants further exploration. While projects such as *TGL* [171] and *DyGLib* [172] have established datasets, models, and evaluation standards for CTDGs, DTDGs suffer from inconsistent ways of dataset preprocessing (e.g., time step segmentation based on the number of edges or the time gap, the train/test split based on the number of edges or snapshots, etc.). In addition, task design and metric selection for anomaly detection and regression tasks require clear standards (e.g., normalization and negative sampling methods) for comparative analysis of the model generalization ability.
2) Exploring and **adopting advanced methods from static graphs** to dynamic graphs is a potential research direction. In addition to architectures such as Graph Transformer, recent techniques such as graph grammatical networks [228], curvature graph neural networks [229], [230], and Unifilter [231] merit attention.
3) With the emergence of applications of DGNNs, the demand for models to provide insights that are understandable to humans has become increasingly important. Although existing methods [232], [233], [234] can provide interpretability for the prediction on static graphs, research on the **interpretability of DGNNs** is still in its early stages [235]. For this reason, finer task definitions and standards are needed. Furthermore, while interpretability studies of static graphs often use synthetic datasets, for dynamic graphs the use of real-world datasets would greatly contribute to the development and practicability of the field.
4) The total number of nodes in different DG datasets varies widely, from small networks with only a few hundred nodes to huge systems with millions of nodes. Meanwhile, the time complexity of graph neural networks varies in a wide range, from sublinear complexity [236] to as high as cubic complexity [36], [37]. Therefore, an in-depth study of the scalability and computational efficiency of DGNNs may not only provide insights for academia but also inspire practical applications in industry. When dealing with huge dynamic graphs, the adoption of distributed systems and learning paradigms [89] can be an effective solution and reveal new research directions.

Although dynamic graph learning is a recent discipline, it will undoubtedly be a major trend for machine learning researchers for years to come. We hope that this review is a modest contribution in this direction.

APPENDIX A STATIC GRAPH ENCODING

For a long time, static graph representation learning has been based on matrix factorization [173], [174], [175], [176], [177] and random walks [132], [133], [143], [178], [179], [180]. Recently, graph neural networks have emerged as powerful approaches, thanks to their ability to learn adaptive parameters to map and filter the signals on the graph. In the following, we split the graph representation learning methods into Random Walk-based, Matrix Factorisation-based, and GNN-based.

A. RANDOM WALK-BASED

After first introduced by Karl in 1905 [181], Random walk-based approaches have then been applied to graph structure sampling. Its variants have been employed for applications including node similarity computation [182], search engines [178], and so on for more than two decades.

A simple example is to randomly sample multiple sequences of nodes by starting from several nodes and walking to their neighboring nodes with probability $1 - a$ or jumping randomly to any node in the graph with probability a. These sequences of nodes can be transformed into attribute vectors of nodes by various methods, such as Skip-Gram model [133] or embedding by neural networks [132], [143]. Similar to **Word2Vec** [180] which generates word embeddings based on word co-occurrence frequencies in natural language processing tasks, **DeepWalk** [132] and **Node2Vec** [143] encode the node embedding according to the sampled node sequences.

B. MATRIX FACTORISATION-BASED

Influenced by the idea of dimensionality reduction, there is a category of Matrix Factorisation-based models [175], [176] [177] in the early research of graph machine learning. For example, **Graph Factorization** [175] encodes the features $\mathbf{x}_i, \mathbf{x}_j$ of nodes i, j as vectors $\mathbf{z}_i, \mathbf{z}_j$ and minimises the difference between their inner product $\langle \mathbf{z}_i, \mathbf{z}_j \rangle$ and their edge weight a_{ij}. Matrix factorization as a kind of shallow embedding method has many limitations: The learned parameters are the embedding of the nodes thus this method is transductive, in addition, matrix factorization is computationally expansive [183].

C. GNN-BASED

Geometric deep learning [27], also known as graph neural networks, has recently emerged as an alternative to traditional methods. Benefiting from the rapid development of the technical and scientific environment: efficient learning algorithms, optimized libraries allowing parallel computing on GPUs, and dedicated hardware, they can iterate over the given inputs and outputs to learn the parameters that encode the input features into low-dimensional informative vectors.

Convolutional neural networks (CNNs) [184] are one of the most popular neural networks because of their capability to weightedly aggregate the neighborhood's features and filter signals. One primary problem with graph neural networks is how to define the convolution on a graph. Different perspectives have led to different implementations: (1) spectral convolution and (2) spatial convolution (or message passing), as well as more complex frameworks like (3) graph Autoencoders and (4) graph Transformers.

From a signal processing point of view, the eigenvectors of the normalized Laplacian matrix of a graph can be considered as the bases of its graph Fourier transform. Multiplying the signal of a transformed node (i.e., a spectrum) in the spectral domain by a signal with learnable parameters (i.e., a convolution kernel) is equivalent to a spectral convolution on the graph. Examples are **Spectral CNN** [185] and its low-order approximation **ChebNet** [29], **GCN** [111].

Otherwise, from the spatial point of view, the convolution on the graph can be regarded as the convolution of the image, but is more complicated due to the irregular grid. Thus spatial convolution can be implemented by message passing between connected nodes e.g. **MPNN** [28], **NN4G** [186], **GraphSage** [9], etc. **GAT** [30] adds the attention mechanism to spatial convolution so that a node aggregates information with different weights according to the attributes of neighboring nodes. Some other approaches incorporate recurrent neural networks and gating mechanisms, such as Gated graph neural network (**GGNN**) [187], combining Gated Recursive Unit (**GRU**) [188] and message passing network, treats message propagation as the recursive update of the hidden states of nodes and controls the update through a gating mechanism.

To encode data $\mathbf{X}$ to a latent representation $\mathbf{Z}$, Autoencoder methods tend to reconstruct $\hat{\mathbf{X}}$ with encoded $\mathbf{Z}$. When getting $\hat{\mathbf{X}}$ and $\mathbf{X}$ as similar as possible, Autoencoder learns the parameters that encode the most informative representation $\mathbf{Z}$. A typical example on graphs, Variational Graph Auto-Encoders (**VGAE**) [189] encodes the feature matrix $\mathbf{X}$ and the adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$ into $\mathbf{Z} \in \mathbb{R}^{|V|\times d_V}$ with a two-layer GCN, and then reconstructs the adjacency matrix $\hat{\mathbf{A}}$ by an inner-product of $\mathbf{Z}$, i.e., $\sigma(\mathbf{Z}.\mathbf{Z}^{\mathrm{T}})$.

In addition to the above models, Transformer models have been applied to graphs in recent years. Min et als. [138] summarizes them as the combinations of the following three approaches: (1) Transformer as auxiliary modules of GNN [190], [191], [192], (2) Use of graph topology for improving self-attention [190], [192], such as using the adjacency matrix as a mask, (3) Use of graph structure during the positional encoding [193], [194], such as degree or spectrum of nodes.

Once the graph is encoded into a latent representation $\mathbf{Z}$, classical tasks such as classification or regression can be performed by a "readout" operation that aggregates information. This can be performed using different pooling methods [195], [196]. A very common case is to use mean pooling to take the average value of the attributes of all nodes as the attributes of the entire graph [27].

Despite the great performance of these model components on various static graph tasks, there are more additional elements to consider when handling dynamic graphs, which are the encoding of temporal and sequential data.

APPENDIX B SEQUENTIAL AND TEMPORAL DATA ENCODING WITH NEURAL NETWORKS

The time information on a dynamic graph may be indexes of the sequence (in STGs and DTDGs) or precise timestamps (in CTDGs), or both. In the literature, neural networks encode sequential or temporal data in different ways, we present the more classical ones in the following.

A. ENCODING SEQUENTIAL DATA

There is a semantic dependency between the elements of sequential data, as in natural language processing tasks. To handle them, the most well-known one is Recurrent Neural Network (**RNN**) [197] which processes each input element recursively along the direction of the sequence. By adding the gating mechanism, Long Short-Term Memory (**LSTM**) [19], [198] and Gated Recursive Unit (**GRU**) [188] have better performance on long sequences.

Recursive networks like RNNs are not suitable for parallel computing, while parallelizable networks based on linear operations and convolution can avoid or reduce the number of sequential operations: Equally using the gating mechanism, the Gated Linear Units (**GLU**) [199] computes the hidden states of the gated input by convolution to avoid dependence on the previous time steps. Some other models use purely convolutional layers, such as causal convolution in **WaveNet** [200], which encodes the signal of the last l time steps through l layers of convolution. When processing long sequences, dilated causal convolution [156], [201] expands the field of perception to avoid too many layers.

Furthermore, **self-attention** [23] modules, better known with the Transformer architecture, can also capture long-term dependencies by using linear transforms to learn the influence of each element in a sequence on each other.

B. ENCODING TIME INFORMATION

Since time t is a numerical value, another practical way is to encode the timestamp or sequence number as a d-dimensional vector as part of the input features.

Common approaches are **Positional Encoding** (PE) [23] which is employed in Transformer architecture, and **Time2Vec** [202] which especially focused on encoding temporal patterns [45]. Both of them use sine or cosine functions, which make the vector representation bounded and smoothly continuous in each dimension, as well as guarantee the translational invariance of time difference values. Another class of methods, Temporal Point Process (**TPP** [130]), models the intensity function $\lambda_i(t)$ of the probability of event i occurring at moment t based on the historical events. Neural networks allow TPP to learn intensity functions automatically to better fit actual situations.

APPENDIX C NOTATIONS USED IN THE ARTICLE

TABLE 9. Commonly used notations.

Notations	Descriptions
$\|\cdot\|$	The length of a set
G	A static graph
DG	A dynamic graph
V	The set of nodes of a graph
E	The set of edges of a graph
T	The set of time steps of a graph
$\mathbf{A}$	The adjacent matrix a graph
$\mathbf{D}$	The degree matrix a graph
u	A certain node $u \in V$
v	A certain node $v \in V$
i, j	The indexes of an event or a node
e	A certain edge $e \in E$
t	Time step / timestamp
t^-	Latest time step / timestamp < t
Δ	Duration
δ	Modification of an edge
d	The dimension of a vector
$\mathbf{X}$	The feature of a (dynamic) graph
$\mathbf{X}_V$	The node feature of a (dynamic) graph
$\mathbf{X}_E$	The edge feature of a (dynamic) graph
$\mathbf{Z}$	The encoded hidden states matrix of a (dynamic) graph
$N_t(v)$	The set of neighboring nodes of node v at time t
NN	A neural network
$f()$	NN for encoding DG
$f_G()$	NN for encoding topological information
$f_T()$	NN for encoding temporal information
θ	Learned model parameters
$\|\|$	Vector concatenation
$\oplus$	Element wise sum
$\odot$	Element wise product

REFERENCES

[1] W. L. Hamilton, "Graph representation learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 14, no. 3, pp. 1–159, 2020.

[2] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," 2020, *arXiv:2006.10637*.

[3] A. Pareja, "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 5363–5370.

[4] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," 2019, *arXiv:1903.08889*.

[5] A. Rehman, M. Ahmad, and O. Khan, "Exploring accelerator and parallel graph algorithmic choices for temporal graphs," in *Proc. 11th Int. Workshop Program. Models Appl. Multicores Manycores*, Feb. 2020, pp. 1–10.

[6] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," 2020, *arXiv:2002.07962*.

[7] B. Bahmani, R. Kumar, M. Mahdian, and E. Upfal, "PageRank on an evolving graph," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2012, pp. 24–32.

[8] C. Song, K. Shu, and B. Wu, "Temporally evolving graph neural network for fake news detection," *Inf. Process. Manage.*, vol. 58, no. 6, Nov. 2021, Art. no. 102712.

[9] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. In Neural Inf. Process. Syst.*, 2017, pp. 1–12.

[10] J. Li, Z. Han, H. Cheng, J. Su, P. Wang, J. Zhang, and L. Pan, "Predicting path failure in time-evolving graphs," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1279–1289.

[11] V. Kalofolias, A. Loukas, D. Thanou, and P. Frossard, "Learning time varying graphs," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 2826–2830.

[12] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," *Int. J. Parallel, Emergent Distrib. Syst.*, vol. 27, no. 5, pp. 387–408, Oct. 2012.

[13] Y. Wang, Y. Yuan, Y. Ma, and G. Wang, "Time-dependent graphs: Definitions, applications, and algorithms," *Data Sci. Eng.*, vol. 4, no. 4, pp. 352–366, Dec. 2019.

[14] H. Huang, A. V. Savkin, and C. Huang, "Reliable path planning for drone delivery using a stochastic time-dependent public transportation network," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 4941–4950, Aug. 2021.

[15] P. Holme and J. Saramäki, "Temporal networks," *Phys. Rep.*, vol. 519, no. 3, pp. 97–125, 2012.

[16] P. Holme, "Modern temporal network theory: A colloquium," *Eur. Phys. J. B*, vol. 88, no. 9, pp. 1–30, Sep. 2015.

[17] Y. Wang, Y.-Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," 2021, *arXiv:2101.05974*.

[18] K. Sato, M. Oka, A. Barrat, and C. Cattuto, "DyANE: Dynamics-aware node embedding for temporal networks," 2019, *arXiv:1909.05976*.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[20] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1003–1012.

[21] P. Hewage, A. Behera, M. Trovati, E. Pereira, M. Ghahremani, F. Palmieri, and Y. Liu, "Temporal convolutional neural (TCN) network for an effective weather forecasting using time-series data from the local weather station," *Soft Comput.*, vol. 24, no. 21, pp. 16453–16482, Nov. 2020.

[22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Montreal, QC, Canada, Dec. 2014, pp. 3104–3112.

[23] A. Vaswani, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–30.

[24] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[25] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, Sep. 2021.

[26] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol. (Short Papers)*, vol. 2, 2018, pp. 464–468.

[27] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[28] J. Gilmer, S. Schoenholz, P. Riley, O. Vinyal, and G. Dahl, "Neural message passing from quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.

[29] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. NIPS*, Barcelona, Spain, 2016, pp. 3844–3852.

[30] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–12.

[31] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Jan. 2020.

[32] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[33] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17.

[34] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," 2020, *arXiv:2003.00982*.

[35] C. Morris, M. Ritzert, M. Fey, W. Hamilton, J. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and lehman go neural: Higher-order graph neural networks," in *Proc. 33rd AAAI Conf. Artif. Intell. 31st Innov. Appl. Artif. Intell. Conf. 9th AAAI Symp. Educ. Adv. Artif. Intell.*, 2019, pp. 1–8, doi: 10.1609/aaai.v33i01.33014602.

[36] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–15.

[37] M. Balcilar, P. Héroux, B. Gauzere, P. Vasseur, S. Adam, and P. Honeine, "Breaking the limits of message passing graph neural networks," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 599–608.

[38] L. Gauvin, A. Panisson, and C. Cattuto, "Detecting the community structure and activity patterns of temporal networks: A non-negative tensor factorization approach," *PLoS ONE*, vol. 9, no. 1, Jan. 2014, Art. no. e86028.

[39] I. Tolstikhin, "MLP-mixer: An all-MLP architecture for vision," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 24261–24272.

[40] D. Li, Q. Lin, and X. Ma, "Identification of dynamic community in temporal network via joint learning graph representation and nonnegative matrix factorization," *Neurocomputing*, vol. 435, pp. 77–90, May 2021.

[41] X. Ma, P. Sun, and Y. Wang, "Graph regularized nonnegative matrix factorization for temporal link prediction in dynamic networks," *Phys. A, Stat. Mech. Appl.*, vol. 496, pp. 121–136, Apr. 2018.

[42] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *Proc. Int. Conf. Neural Inf. Process.* Cham, Switzerland: Springer, 2018, pp. 362–373.

[43] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–25.

[44] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *J. Mach. Learn. Res.*, vol. 21, no. 70, pp. 1–73, 2020.

[45] J. Skarding, B. Gabrys, and K. Musial, "Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey," *IEEE Access*, vol. 9, pp. 79143–79168, 2021.

[46] C. D. T. Barros, M. R. F. Mendonça, A. B. Vieira, and A. Ziviani, "A survey on embedding dynamic graphs," *ACM Comput. Surveys*, vol. 55, no. 1, pp. 1–37, Jan. 2023.

[47] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini, "Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities," 2023, *arXiv:2302.01018*.

[48] S. Ma, J.-W. Liu, X. Zuo, and W.-M. Li, "Heterogeneous graph gated attention network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–6.

[49] W. Luo, H. Zhang, X. Yang, L. Bo, X. Yang, Z. Li, X. Qie, and J. Ye, "Dynamic heterogeneous graph neural network for real-time event prediction," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 3213–3223.

[50] A. Zaki, M. Attia, D. Hegazy, and S. Amin, "Comprehensive survey on dynamic graph models," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 2, pp. 1–10, 2016.

[51] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph WaveNet for deep spatial–temporal graph modeling," 2019, *arXiv:1906.00121*.

[52] J. Gao and B. Ribeiro, "On the equivalence between temporal and static equivariant graph representations," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 7052–7076.

[53] J. Guo, Z. Han, Z. Su, J. Li, V. Tresp, and Y. Wang, "Continuous temporal graph networks for event-based graph data," 2022, *arXiv:2205.15924*.

[54] A. McGregor, "Graph stream algorithms: A survey," *ACM SIGMOD Rec.*, vol. 43, no. 1, pp. 9–20, May 2014.

[55] J. Zhang, "A survey on streaming algorithms for massive graphs," in *Managing and Mining Graph Data*, 2010, pp. 393–420.

[56] M. Azimifar, T. D. Todd, A. Khezrian, and G. Karakostas, "Vehicle-to-vehicle forwarding in green roadside infrastructure," *IEEE Trans. Veh. Technol.*, vol. 65, no. 2, pp. 780–795, Feb. 2016.

[57] J. Li, P. Wang, H. Li, and K. Shi, "Enhanced time-expanded graph for space information network modeling," *Sci. China Inf. Sci.*, vol. 65, no. 9, Sep. 2022, Art. no. 192301.

[58] O. Michail, "An introduction to temporal graphs: An algorithmic perspective," 2015, *arXiv:1503.00278*.

[59] X. Yang, Z. Song, I. King, and Z. Xu, "A survey on deep semi-supervised learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 9, pp. 8934–8954, Sep. 2023.

[60] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," 2017, *arXiv:1709.04875*.

[61] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," 2017, *arXiv:1707.01926*.

[62] L. Ruiz, F. Gama, and A. Ribeiro, "Gated graph recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 6303–6318, 2020.

[63] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 753–763.

[64] S. Deng, S. Wang, H. Rangwala, L. Wang, and Y. Ning, "Cola-GNN: Cross-location attention based graph neural networks for long-term ILI prediction," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2020, pp. 245–254.

[65] A. Kapoor, X. Ben, L. Liu, B. Perozzi, M. Barnes, M. Blais, and S. O'Banion, "Examining COVID-19 forecasting using spatio-temporal graph neural networks," 2020, *arXiv:2007.03113*.

[66] L. Wang, A. Adiga, J. Chen, A. Sadilek, S. Venkatramanan, and M. Marathe, "CausalGNN: Causal-based graph neural networks for spatio-temporal epidemic forecasting," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 11, Jun. 2022, pp. 12191–12199.

[67] L. Xia, C. Huang, Y. Xu, P. Dai, L. Bo, X. Zhang, and T. Chen, "Spatial–temporal sequential hypergraph network for crime prediction with dynamic multiplex relation learning," 2022, *arXiv:2201.02435*.

[68] J. Fan, J. Bai, Z. Li, A. Ortiz-Bobea, and C. Gomes, "A GNN-RNN approach for harnessing geospatial and temporal information: Application to crop yield prediction," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 11873–11881.

[69] Z. Jia, Y. Lin, J. Wang, R. Zhou, X. Ning, Y. He, and Y. Zhao, "GraphSleepNet: Adaptive spatial–temporal graph convolutional networks for sleep stage classification," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 1324–1330.

[70] L. Chaolong, C. Zhen, Z. Wenming, X. Chunyan, and Y. Jian, "Spatio-temporal graph convolution for skeleton based action recognition," in *Proc. Irty-2nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–4.

[71] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12012–12038, Dec. 2023.

[72] A. Behrouz and M. Seltzer, "Anomaly detection in multiplex dynamic networks: From blockchain security to brain disease prediction," 2022, *arXiv:2211.08378*.

[73] Y. Liu, S. Pan, Y. G. Wang, F. Xiong, L. Wang, Q. Chen, and V. C. Lee, "Anomaly detection in dynamic graphs via transformer," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12081–12094, Dec. 2023.

[74] L. Cai, "Structural temporal graph neural networks for anomaly detection in dynamic graphs," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, Oct. 2021, pp. 3747–3756.

[75] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "NetWalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 2672–2681.

[76] L. Zheng, Z. Li, J. Li, Z. Li, and J. Gao, "AddGraph: Anomaly detection in dynamic graph using attention-based temporal GCN," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, p. 7.

[77] Y. Zhou, W. Liu, Y. Pei, L. Wang, D. Zha, and T. Fu, "Dynamic network embedding by semantic evolution," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.

[78] J. Choi, T. Ko, Y. Choi, H. Byun, and C.-K. Kim, "Dynamic graph convolutional networks with attention mechanism for rumor detection on social media," *PLoS ONE*, vol. 16, no. 8, Aug. 2021, Art. no. e0256039.

[79] M. Sun, X. Zhang, J. Zheng, and G. Ma, "DDGCN: Dual dynamic graph convolutional networks for rumor detection on social media," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 4, Jun. 2022, pp. 4611–4619.

[80] H. Tang, G. Zhao, X. Bu, and X. Qian, "Dynamic evolution of multi-graph based collaborative filtering for recommendation systems," *Knowl.-Based Syst.*, vol. 228, Sep. 2021, Art. no. 107251.

[81] X. Li, M. Zhang, S. Wu, Z. Liu, L. Wang, and P. S. Yu, "Dynamic graph collaborative filtering," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2020, pp. 322–331.

[82] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1269–1278.

[83] L. Qu, H. Zhu, Q. Duan, and Y. Shi, "Continuous-time link prediction via temporal dependent graph neural network," in *Proc. Web Conf.*, Apr. 2020, pp. 3026–3032.

[84] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Proc. Companion Web Conf. Web Conf. (WWW)*, 2018, pp. 969–976.

[85] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 2017, *arXiv:1709.05584*.

[86] H. Cai, V. W. Zheng, and K. C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1616–1637, Sep. 2018.

[87] G. Xue, M. Zhong, J. Li, J. Chen, C. Zhai, and R. Kong, "Dynamic network embedding survey," *Neurocomputing*, vol. 472, pp. 212–223, Feb. 2022.

[88] Y. Wang, P. Li, C. Bai, V. Subrahmanian, and J. Leskovec, "Generic representation learning for dynamic social interaction," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining Workshop*, 2020, pp. 1–9.

[89] I. King and H. Huang, "Euler: Detecting network lateral movement via scalable temporal link prediction," *ACM Trans. Privacy Secur.*, vol. 26, no. 3, pp. 1–36, 2023.

[90] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognit.*, vol. 97, Jan. 2020, Art. no. 107000.

[91] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5308–5317.

[92] M. Jiang, J. Dong, D. Ma, J. Sun, J. He, and L. Lang, "Inception spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. Int. Symp. Control Eng. Robot. (ISCER)*, Feb. 2022, pp. 208–213.

[93] Z. Ding, Y. Ma, B. He, and V. Tresp, "A simple but powerful graph encoder for temporal knowledge graph completion," 2021, *arXiv:2112.07791*.

[94] S. Khoshraftar, S. Mahdavi, A. An, Y. Hu, and J. Liu, "Dynamic graph embedding via LSTM history tracking," in *Proc. IEEE Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2019, pp. 119–127.

[95] D. Lin, J. Wu, Q. Yuan, and Z. Zheng, "T-EDGE: Temporal WEighted MultiDiGraph embedding for Ethereum transaction network analysis," *Frontiers Phys.*, vol. 8, p. 204, Jun. 2020.

[96] C. Song, Y. Lin, S. Guo, and H. Wan, "Spatial–temporal synchronous graph convolutional networks: A new framework for spatial–temporal network data forecasting," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 914–921.

[97] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Node classification in temporal graphs through stochastic sparsification and temporal structural convolution," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, Ghent, Belgium, Sep. 2020, pp. 330–346.

[98] J. Grigsby, Z. Wang, N. Nguyen, and Y. Qi, "Long-range transformers for dynamic spatiotemporal forecasting," 2021, *arXiv:2109.12218*.

[99] A. Feng and L. Tassiulas, "Adaptive graph spatial–temporal transformer network for traffic flow forecasting," 2022, *arXiv:2207.05064*.

[100] S. Banerjee, M. Dong, and W. Shi, "Spatial–temporal synchronous graph transformer network (STSGT) for COVID-19 forecasting," *Smart Health*, vol. 26, Dec. 2022, Art. no. 100348.

[101] G. Zheng, W. K. Chai, J. Zhang, and V. Katos, "VDGCNeT: A novel network-wide virtual dynamic graph convolution neural network and transformer-based traffic prediction model," *Knowl.-Based Syst.*, vol. 275, Sep. 2023, Art. no. 110676.

[102] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial–temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, 2019, pp. 922–929.

[103] D. Xu, W. Cheng, D. Luo, X. Liu, and X. Zhang, "Spatio-temporal attentive RNN for node classification in temporal attributed graphs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 3947–3953.

[104] Y. Fan, M. Ju, C. Zhang, and Y. Ye, "Heterogeneous temporal graph neural network," in *Proc. SIAM Int. Conf. Data Mining (SDM)*, 2022, pp. 657–665.

[105] J. Chen, X. Wang, and X. Xu, "GC-LSTM: Graph convolution embedded LSTM for dynamic link prediction," 2018, *arXiv:1812.04206*.

[106] A. Nicolicioiu, I. Duta, and M. Leordeanu, "Recurrent space-time graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 12838–12850.

[107] I. Duta, A. Nicolicioiu, and M. Leordeanu, "Dynamic regions graph neural networks for spatio-temporal reasoning," in *Proc. ORLR Workshop, Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 1–13.

[108] B. Kim, J. Ye, and J. Kim, "Learning dynamic graph representation of brain connectome with spatio-temporal attention," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 4314–4327.

[109] L. Yang, Z. Xiao, W. Jiang, Y. Wei, Y. Hu, and H. Wang, "Dynamic heterogeneous graph embedding using hierarchical attentions," in *Proc. Eur. Conf. Inf. Retr.* Cham, Switzerland: Springer, 2020, pp. 425–432.

[110] J. Chen, J. Zhang, X. Xu, C. Fu, D. Zhang, Q. Zhang, and Q. Xuan, "E-LSTM-D: A deep learning framework for dynamic network link prediction," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 51, no. 6, pp. 3699–3712, Jun. 2021.

[111] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[112] S.-H. Teng, "Scalable algorithms for data and network analysis," *Found. Trends Theor. Comput. Sci.*, vol. 12, nos. 1–2, pp. 1–274, 2016.

[113] T. Bian, X. Xiao, T. Xu, P. Zhao, W. Huang, Y. Rong, and J. Huang, "Rumor detection on social media with bi-directional graph convolutional networks," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 549–556.

[114] P. Goyal, N. Kamra, X. He, and Y. Liu, "DynGEM: Deep embedding method for dynamic graphs," 2018, *arXiv:1805.11273*.

[115] E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou, and X. Qian, "Variational graph recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–2.

[116] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "DySAT: Deep neural representation learning on dynamic graphs via self-attention networks," in *Proc. 13th Int. Conf. Web Search Data Mining*, Jan. 2020, pp. 519–527.

[117] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70. Sydney, NSW, Australia: International Convention Centre, Aug. 2017, pp. 1243–1252.

[118] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Self-attention with functional time representation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, 32.

[119] X. Chang, X. Liu, J. Wen, S. Li, Y. Fang, L. Song, and Y. Qi, "Continuous-time dynamic graph learning via neural interaction processes," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2020, pp. 145–154.

[120] B. Kim, J. Choi, E. Yun, K. Kim, X. Li, and J. Lee, "Large-scale graph representation learning of dynamic brain connectome with transformers," in *Proc. Temporal Graph Learn. Workshop@ NeurIPS*, 2023, pp. 1–7.

[121] A.-T. Kuo, H. Chen, Y.-H. Kuo, and W.-S. Ku, "Dynamic graph representation learning for depression screening with transformer," 2023, *arXiv:2305.06447*.

[122] W. Cong, Y. Wu, Y. Tian, M. Gu, Y. Xia, M. Mahdavi, and C. Chen, "Dynamic graph representation learning via graph transformer networks," in *Proc. ICLR*, 2021, pp. 1–11.

[123] S. Wei, B. Wu, A. Xiang, Y. Zhu, and C. Song, "DGTR: Dynamic graph transformer for rumor detection," *Frontiers Res. Metrics Anal.*, vol. 7, Jan. 2023, Art. no. 1055348.

[124] L. Xia, C. Huang, Y. Xu, and J. Pei, "Multi-behavior sequential recommendation with temporal graph transformer," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 6, pp. 6099–6112, Jun. 2023.

[125] Y. Fan, M. Ju, S. Hou, Y. Ye, W. Wan, K. Wang, Y. Mei, and Q. Xiong, "Heterogeneous temporal graph transformer: An intelligent system for evolving Android malware detection," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2021, pp. 2831–2839.

[126] M. Biparva, R. Karimi, F. Faez, and Y. Zhang, "TodyFormer: Towards holistic dynamic graph transformers with structure-aware tokenization," in *Proc. Temporal Graph Learn. Workshop@ NeurIPS 2023*, 2023.

[127] L. Wang, X. Chang, S. Li, Y. Chu, H. Li, W. Zhang, X. He, L. Song, J. Zhou, and H. Yang, "TCL: Transformer-based dynamic graph modelling via contrastive learning," 2021, *arXiv:2105.07944*.

[128] X. Wang, D. Lyu, M. Li, Y. Xia, Q. Yang, X. Wang, X. Wang, P. Cui, Y. Yang, B. Sun, and Z. Guo, "APAN: Asynchronous propagation attention network for real-time temporal graph embedding," in *Proc. Int. Conf. Manage. Data*, Jun. 2021, pp. 2628–2638.

[129] W. Cong, S. Zhang, J. Kang, B. Yuan, H. Wu, X. Zhou, H. Tong, and M. Mahdavi, "Do we really need complicated model architectures for temporal networks?" 2023, *arXiv:2302.11636*.

[130] P. Lewis, "Multivariate point processes," in *Proc. Berkeley Symp. Math. Statist. Probab.*, 1972, p. 401.

[131] C. Huang, L. Wang, X. Cao, W. Ma, and S. Vosoughi, "Learning dynamic graph embeddings using random walk with temporal backtracking," in *Proc. NeurIPS Temporal Graph Learn. Workshop*, 2022.

[132] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data mining*, Aug. 2014, pp. 701–710.
[133] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proc. 24th Int. Conf. World Wide Web*, May 2015, pp. 1067–1077.
[134] M. Jin, Y. Li, and S. Pan, "Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 19874–19886.
[135] A. Behrouz, F. Hashemi, S. Sadeghian, and M. Seltzer, "CAT-walk: Inductive hypergraph learning via set walks," 2023, *arXiv:2306.11147*.
[136] H. Li, S. Zhang, X. Li, L. Su, H. Huang, D. Jin, L. Chen, J. Huang, and J. Yoo, "DetectorNet: Transformer-enhanced spatial temporal graph neural network for traffic prediction," in *Proc. 29th Int. Conf. Adv. Geographic Inf. Syst.*, Nov. 2021, pp. 133–136.
[137] L. Müller, M. Galkin, C. Morris, and L. Rampásek, "Attending to graph transformers," 2023, *arXiv:2302.04181*.
[138] E. Min, R. Chen, Y. Bian, T. Xu, K. Zhao, W. Huang, P. Zhao, J. Huang, S. Ananiadou, and Y. Rong, "Transformer for graphs: An overview from architecture perspective," 2022, *arXiv:2202.08455*.
[139] S. Hu, G. Zou, S. Lin, L. Wu, C. Zhou, B. Zhang, and Y. Chen, "Recurrent transformer for dynamic graph representation learning with edge temporal states," 2023, *arXiv:2304.10079*.
[140] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 135–144.
[141] H. Huang, R. Shi, W. Zhou, X. Wang, H. Jin, and X. Fu, "Temporal heterogeneous information network embedding," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, Aug. 2021, pp. 1470–1476.
[142] F. Zhou, X. Xu, C. Li, G. Trajcevski, T. Zhong, and K. Zhang, "A heterogeneous dynamical graph neural networks approach to quantify scientific impact," 2020, *arXiv:2003.12042*.
[143] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864.
[144] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1225–1234.
[145] L. Zhao and L. Akoglu, "PairNorm: Tackling oversmoothing in GNNs," 2019, *arXiv:1909.12223*.
[146] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 4, 2020, pp. 3438–3445.
[147] C. Cai and Y. Wang, "A note on over-smoothing for graph neural networks," 2020, *arXiv:2006.13318*.
[148] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," 2020, *arXiv:2006.05205*.
[149] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," 2021, *arXiv:2111.14522*.
[150] F. Di Giovanni, L. Giusti, F. Barbero, G. Luise, P. Lio', and M. Bronstein, "On over-squashing in message passing neural networks: The impact of width, depth, and topology," 2023, *arXiv:2302.02941*.
[151] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein, "Robust optimization as data augmentation for large-scale graphs," 2020, *arXiv:2010.09891*.
[152] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah, "Data augmentation for graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 12, 2021, pp. 11015–11023.
[153] Y. Wang, Y. Cai, Y. Liang, H. Ding, C. Wang, S. Bhatia, and B. Hooi, "Adaptive data augmentation on temporal graphs," in *Proc. Adv. In Neural Inf. Process. Syst.*, 2021, pp. 1440–1452.
[154] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.
[155] E. Tiukhova, E. Penaloza, M. Óskarsdóttir, H. Garcia, A. C. Bahnsen, B. Baesens, M. Snoeck, and C. Bravo, "Influencer detection with dynamic graph neural networks," 2022, *arXiv:2211.09664*.
[156] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, *arXiv:1511.07122*.
[157] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, 2016.
[158] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.
[159] J. Ma, W. Gao, P. Mitra, S. Kwon, B. Jansen, K. Wong, and M. Cha, *Detecting Rumors From Microblogs With Recurrent Neural Networks*. Washington, DC, USA: Proc. AAAI Press, 2016.
[160] X. Liu, A. Nourbakhsh, Q. Li, R. Fang, and S. Shah, "Real-time rumor debunking on Twitter," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2015, pp. 1867–1870.
[161] J. Ma, W. Gao, and K.-F. Wong, "Detect rumors in microblog posts using propagation structure via kernel learning," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics (Long Papers)*, vol. 1, 2017, pp. 1–10.
[162] K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu, "FakeNewsNet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media," *Big Data*, vol. 8, no. 3, pp. 171–188, Jun. 2020.
[163] A. Zubiaga, M. Liakata, and R. Procter, "Exploiting context for rumour detection in social media," in *Social Informatics* (Lecture Notes in Computer Science), vol. 10539. Cham, Switzerland: Springer, 2017, pp. 109–123.
[164] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A large scale dataset for 3D human activity analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1010–1019.
[165] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, and A. C. Kot, "NTU RGB+D 120: A large-scale benchmark for 3D human activity understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 10, pp. 2684–2701, Oct. 2020.
[166] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber, *Documentation Mocap Database HDM05*. Bonn, Germany: Universität Bonn, 2007.
[167] C. O'Reilly, N. Gosselin, J. Carrier, and T. Nielsen, "Montreal archive of sleep studies: An open-access resource for instrument benchmarking and exploratory research," *J. Sleep Res.*, vol. 23, no. 6, pp. 628–635, Dec. 2014.
[168] J. Leskovec and A. Krevl. (Jun. 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. [Online]. Available: http://snap.stanford.edu/data
[169] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. AAAI*, 2015, pp. 1–2. [Online]. Available: https://networkrepository.com
[170] B. Klimt and Y. Yang, "The Enron corpus: A new dataset for email classification research," in *Proc. Eur. Conf. Mach. Learn.*, 2004, pp. 217–226.
[171] S. Huang, F. Poursafaei, J. Danovitch, M. Fey, W. Hu, E. Rossi, J. Leskovec, M. Bronstein, G. Rabusseau, and R. Rabbany, "Temporal graph benchmark for machine learning on temporal graphs," 2023, *arXiv:2307.01026*.
[172] L. Yu, L. Sun, B. Du, and W. Lv, "Towards better dynamic graph learning: New architecture and unified library," 2023, *arXiv:2303.13047*.
[173] J. Yang, S. Yang, Y. Fu, X. Li, and T. Huang, "Non-negative graph embedding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2008, pp. 1–8.
[174] S. Sarkar and A. Dong, "Community detection in graphs using singular value decomposition," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 83, Apr. 2011, Art. no. 046114.
[175] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proc. 22nd Int. Conf. World Wide Web*, May 2013, pp. 37–48.
[176] S. Cao, W. Lu, and Q. Xu, "GraRep: Learning graph representations with global structural information," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2015, pp. 891–900.
[177] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1105–1114.
[178] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*. Stanford, CA, USA: Stanford InfoLab, 1999.
[179] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2002, pp. 538–543.
[180] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.

[181] K. Pearson, "The problem of the random walk," *Nature*, vol. 72, p. 342, Aug. 1905.

[182] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Proc. 6th Int. Conf. Data Mining (ICDM)*, Dec. 2006, pp. 613–622.

[183] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," 2020, *arXiv:2005.03675*.

[184] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Netw.*, 1995, p. 3361.

[185] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*.

[186] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Trans. Neural Netw.*, vol. 20, no. 3, pp. 498–511, Mar. 2009.

[187] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2015, *arXiv:1511.05493*.

[188] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.

[189] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016, *arXiv:1611.07308*.

[190] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T. Liu, "Do transformers really perform badly for graph representation?" in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 28877–28888.

[191] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang, "Self-supervised graph transformer on large-scale molecular data," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. –12571.

[192] J. Zhang, H. Zhang, C. Xia, and L. Sun, "Graph-bert: Only attention is needed for learning graph representations," 2020, *arXiv:2001.05140*.

[193] J. Zhao, C. Li, Q. Wen, Y. Wang, Y. Liu, H. Sun, X. Xie, and Y. Ye, "Gophormer: Ego-graph transformer for node classification," 2021, *arXiv:2110.13094*.

[194] P. Thölke and G. De Fabritiis, "TorchMD-NET: Equivariant transformers for neural network based molecular potentials," 2022, *arXiv:2202.02541*.

[195] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1–12.

[196] H. Gao, Y. Liu, and S. Ji, "Topology-aware graph pooling networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 12, pp. 4512–4518, Dec. 2021.

[197] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[198] F. Gers, "Long short-term memory in recurrent neural networks," in *Verlag Nicht Ermittelbar*, 2001.

[199] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 933–941.

[200] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," 2016, *arXiv:1609.03499*.

[201] Y.-F. Zhang, P. J. Thorburn, and P. Fitch, "Multi-task temporal convolutional network for predicting water quality sensor data," in *Proc. Int. Conf. Neural Inf. Process.*, Sydney, NSW, Australia, Dec. 2019, pp. 122–130.

[202] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker, "Time2 Vec: Learning a vector representation of time," 2019, *arXiv:1907.05321*.

[203] D. Wang, Z. Zhang, Y. Ma, T. Zhao, T. Jiang, N. V. Chawla, and M. Jiang, "Modeling co-evolution of attributed and structural information in graph sequence," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 2, pp. 1817–1830, Feb. 2023.

[204] L. Yang, C. Chatelain, and S. Adam, "DspGNN: Bringing spectral design to discrete time dynamic graph neural networks for edge regression," *Proc. Temporal Graph Learn. Workshop@ NeurIPS*, 2023.

[205] L. Jiang, C. Zhang, F. Poursafaei, and S. Huang, "Towards temporal edge regression: A case study on agriculture trade between nations," 2023, *arXiv:2308.07883*.

[206] S. Kumar, F. Spezzano, V. S. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 221–230.

[207] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. S. Subrahmanian, "REV2: Fraudulent user prediction in rating platforms," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, Feb. 2018, pp. 333–341.

[208] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2005, pp. 177–187.

[209] S. Kumar, C. Bai, V. Subrahmanian, and J. Leskovec, "Deception detection in group video conversations using dynamic interaction networks," in *Proc. ICWSM*, 2021, pp. 339–350.

[210] C. Bai, S. Kumar, J. Leskovec, M. Metzger, J. F. Nunamaker, and V. S. Subrahmanian, "Predicting the visual focus of attention in multi-person discussion videos," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 1–7.

[211] P. Panzarasa, T. Opsahl, and K. M. Carley, "Patterns and dynamics of users' behavior and interaction: Network analysis of an online community," *J. Amer. Soc. Inf. Sci. Technol.*, vol. 60, no. 5, pp. 911–932, May 2009.

[212] J. Gehrke, P. Ginsparg, and J. Kleinberg, "Overview of the 2003 KDD cup," *ACM SIGKDD Explor. Newslett.*, vol. 5, no. 2, pp. 149–151, Dec. 2003.

[213] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. ACM SIGKDD Workshop Mining Data Semantics*, 2012, pp. 1–8.

[214] J. Kunegis, "KONECT: The Koblenz network collection," in *Proc. 22nd Int. Conf. World Wide Web*, May 2013, pp. 1343–1350. [Online]. Available: http://dl.acm.org/citation.cfm?id=2488173

[215] T. Nie, G. Qin, W. Ma, Y. Mei, and J. Sun, "ImputeFormer: Low rankness-induced transformers for generalizable spatiotemporal imputation," 2023, *arXiv:2312.01728*.

[216] H. Lin, X. Zhang, and X. Fu, "A graph convolutional encoder and decoder model for rumor detection," in *Proc. IEEE 7th Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2020, pp. 300–306.

[217] M. Rahman and M. Hasan, "Link prediction in dynamic networks using graphlet," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, Riva Del Garda, Italy, Sep. 2016, pp. 394–409.

[218] M. Weber, G. Domeniconi, J. Chen, D. Karl I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson, "Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics," 2019, *arXiv:1908.02591*.

[219] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proc. 42nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2019, pp. 165–174.

[220] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *Proc. 2nd ACM Workshop Online Social Netw.*, Aug. 2009, pp. 37–42.

[221] M. De Choudhury, H. Sundaram, A. John, and D. D. Seligmann, "Social synchrony: Predicting mimicry of user actions in online social media," in *Proc. Int. Conf. Comput. Sci. Eng.*, 2009, pp. 151–158.

[222] P. H. Schönemann, "A generalized solution of the orthogonal Procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, Mar. 1966.

[223] S. Agarwal, R. Sawhney, M. Thakkar, P. Nakov, J. Han, and T. Derr, "THINK: Temporal hypergraph hyperbolic network," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2022, pp. 849–854.

[224] T. Opsahl and P. Panzarasa, "Clustering in weighted networks," *Social Netw.*, vol. 31, no. 2, pp. 155–163, May 2009.

[225] A. Madan, M. Cebrian, S. Moturu, and K. Farrahi, "Sensing the 'health state' of a community," *IEEE Pervasive Comput.*, vol. 11, no. 4, pp. 36–45, Oct./Dec. 2011.

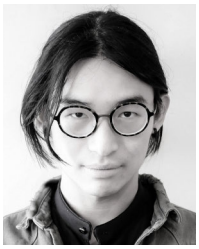
[226] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Commun. ACM*, vol. 57, no. 7, pp. 86–94, Jul. 2014.

[227] X. Bresson and T. Laurent, "Residual gated graph ConvNets," 2017, *arXiv:1711.07553*.

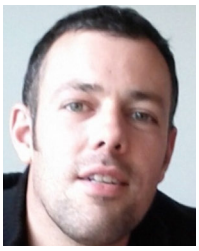
[228] J. Piquenot, A. Moscatelli, M. Berar, P. Héroux, R. Raveaux, J. Ramel, and S. Adam, "G^2N^2: Weisfeiler and Lehman go grammatical," in *Proc. 12th Int. Conf. Learn. Represent.*, 2024, pp. 1–9.

[229] H. Li, J. Cao, J. Zhu, Y. Liu, Q. Zhu, and G. Wu, "Curvature graph neural network," *Inf. Sci.*, vol. 592, pp. 50–66, May 2022.

[230] H. Attali, D. Buscaldi, and N. Pernelle, "Matrice d'adjacence courbée: Intégration de la courbure des arêtes dans la transmission des messages," in *Revue Des Nouvelles Technologies De L'Information*. Paris, France: Extraction et Gestion des Connaissances, 2004, pp. 71–82.
[231] K. Huang and P. Liò, "An effective universal polynomial basis for spectral graph neural networks," 2023, *arXiv:2311.18177*.
[232] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNexplainer: Generating explanations for graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–8.
[233] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 19620–19631.
[234] W. Lin, H. Lan, H. Wang, and B. Li, "OrphicX: A causality-inspired latent variable model for interpreting graph neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 13719–13728.
[235] K. Zhao and L. Zhang, "Causality-inspired spatial–temporal explanations for dynamic graph neural networks," in *Proc. 12th Int. Conf. Learn. Represent.*, 2024.
[236] M. Ding, T. Rabbani, B. An, E. Wang, and F. Huang, "Sketch-GNN: Scalable graph neural networks with sublinear training complexity," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 2930–2943.



LESHANSHUI YANG received the M.S. degree in data science from the University of Rouen Normandy, France, in 2020. He is currently pursuing the joint Ph.D. degree in dynamic graph neural networks with Saagie and Laboratoire d'Informatique, Traitement de l'information et des Systmes (LITIS), France. His research interests include dynamic graph embedding, graph transformers, and the interpretability of graph neural networks.



CLÉMENT CHATELAIN received the Ph.D. degree in machine learning for handwriting recognition from the University of Rouen, in 2006. Since 2007, he has been an Associate Professor with INSA Rouen Normandy and LITIS Laboratory. His research interests include machine learning on sequence and graphs, with application on document image analysis and medical imaging.

SÉBASTIEN ADAM received the Ph.D. degree in graphical document analysis, in 2001. He is currently a Full Professor with the LITIS Laboratory, Rouen, France. His research interests include machine learning, with a focus on geometric deep learning through contributions about spectral GNNs, expressive power of GNNs, and dynamic GNNs, with application on chemistry and sport analysis.