A Survey on Graph Representation Learning Methods

SHIMA KHOSHRAFTAR and AIJUN AN, Electrical Engineering and Computer Science Department, York University, Canada

Graph representation learning has been a very active research area in recent years. The goal of graph representation learning is to generate graph representation vectors that capture the structure and features of large graphs accurately. This is especially important because the quality of the graph representation vectors will affect the performance of these vectors in downstream tasks such as node classification, link prediction and anomaly detection. Many techniques have been proposed for generating effective graph representation vectors, which generally fall into two categories: traditional graph embedding methods and graph neural network (GNN)–based methods. These methods can be applied to both static and dynamic graphs. A static graph is a single fixed graph, whereas a dynamic graph evolves over time and its nodes and edges can be added or deleted from the graph. In this survey, we review the graph-embedding methods in both traditional and GNN-based categories for both static and dynamic graphs and include the recent papers published until the time of submission. In addition, we summarize a number of limitations of GNNs and the proposed solutions to these limitations. Such a summary has not been provided in previous surveys. Finally, we explore some open and ongoing research directions for future work.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **neural networks**; **learning latent representations;**

Additional Key Words and Phrases: Graphs, graph representation learning, graph neural network, graph embedding

ACM Reference format:
Shima Khoshraftar and Aijun An. 2024. A Survey on Graph Representation Learning Methods. *ACM Trans. Intell. Syst. Technol.* 15, 1, Article 19 (January 2024), 55 pages.
https://doi.org/10.1145/3633518

1 INTRODUCTION

Graphs are powerful data structures to represent networks that contain entities and relationships between entities. There are very large networks in different domains, including social networks, financial transactions, and biological networks. For instance, in social networks, people are the nodes and their friendships constitute the edges. In financial transactions, the nodes and edges could be people and their money transactions. In biological networks, nodes are biological entities such as proteins and genes and edges are the interactions between them. One of the strengths of a graph data structure is its generality, meaning that the same structure can be used to represent

Authors' address: S. Khoshraftar and A. An, Electrical Engineering and Computer Science Department, York University, Keele Street, Toronto, Canada; e-mails: {khoshraf, aan}@eecs.yorku.ca.

2157-6904/2024/01-ART19 $15.00
https://doi.org/10.1145/3633518

different networks. In addition, graphs have strong foundations in mathematics, which could be leveraged for analyzing and learning from complex networks.

In order to use graphs in different downstream applications, it is important to represent them effectively. A graph can be simply represented using the adjacency matrix, which is a square matrix whose elements indicate whether pairs of vertices are adjacent or not in the graph, or using the extracted features of the graph. However, the dimensionality of the adjacency matrix is often very high for big graphs, and the feature extraction–based methods are time-consuming and may not represent all the necessary information in the graphs. Recently, the abundance of data and computation resources have paved the way for more flexible graph representation methods. Specifically, graph embedding methods have been very successful in graph representation. These methods project the graph elements (such as nodes, edges, and subgraphs) to a lower dimensional space and preserve the properties of graphs. Graph embedding methods[1] can be categorized into traditional graph embedding and **graph neural network (GNN)**–based graph embedding methods. Traditional graph embedding methods capture the information in a graph by applying different techniques, including random walks, factorization methods, and non-GNN–based deep learning. These methods can be applied to both static and dynamic graphs. A static graph is a single fixed graph, whereas a dynamic graph evolves over time and its nodes and edges can be added or deleted from the graph. For example, a molecule can be represented as a static graph, whereas a social network can be represented by a dynamic graph. GNNs are another category of graph embedding methods that have been proposed recently. In GNNs, node embeddings are obtained by aggregating the embeddings of the node's neighbors. Early works on GNNs were based on recurrent neural networks. However, later convolutional GNNs were developed that are based on the convolution operation. In addition, there are spatial-temporal GNNs and dynamic GNNs that leverage the strengths of GNNs in evolving networks.

In this survey, we conduct a review of both traditional and GNN-based graph embedding methods in static and dynamic settings. To the best of our knowledge, there are several other surveys and books on graph representation learning. The surveys in [9, 29, 38, 90, 100, 101, 151, 181, 201, 295, 306, 341, 357, 369, 370] mainly cover the graph embedding methods for static graphs and their coverage of dynamic graph embedding methods is limited, if any. Inversely, [14, 136, 252, 298] surveys mainly focus on dynamic graph embedding methods. [84, 293] review both the static and dynamic graph embedding methods but they focus on GNN-based methods. The distinctions between our survey and others are as follows:

(1) We put together the graph embedding methods in both traditional and GNN-based categories for both static and dynamic graphs and include over 300 papers published in reputable venues in data mining, machine learning, and artificial intelligence [2] since 2017 until the time of this submission, as well as influential papers with high citations published before 2017.
(2) We summarize a number of limitations of GNN-based methods and the proposed solutions to these limitations until the time of submission. These limitations are expressive power, oversmoothing, scalability, over-squashing, capturing long-range dependencies, design space, neglecting substructures, homophily assumptions, and catastrophic forgetting. Such a summary was not provided in previous surveys.
(3) We provide a list of the real-world applications of GNN-based methods that are deployed in production.

[1]The methods covered in the survey do not consider or distinguish the types of nodes and/or edges. There are other surveys that take into account the heterogeneity of graph structure [278, 309].
[2]The venues include KDD, ICLR, ICML, NeurIPS, AAAI, IJCAI, ICDM, WWW, WSDM, DSAA, SDM, and CIKM.

(4) We suggest a list of future research directions, including new ones that are not covered by previous surveys.

The organization of the survey is as follows. In Section 2, we provide basic background on graphs. In Section 3, the traditional node embedding methods for static and dynamic graphs are reviewed. In Section 4, we survey static, spatial-temporal and dynamic GNN-based graph embedding methods and their real-world applications, limitations, and implementation libraries. In Section 5, we provide a list of some common datasets used for evaluating graph embedding methods. In Section 6, we describe some embedding methods that are specially designed for heterogeneous graphs, bipartite graphs, and hypergraphs, which were not covered in the earlier sections. Finally, in Section 7, we discuss the ongoing and future research directions in the graph representation learning area.

2 GRAPHS

Graphs are powerful tools for representing entities and relationships between them. Graphs have applications in many domains, including social networks, E-commerce, and citation networks. In social networks such as Facebook, nodes in the graph are the people and the edges represent the friendship between them. In E-commerce, the Amazon network is a good example, in which users and items are the nodes and the buying or selling relationships are the edges.

Definition 1. Formally, a graph G is defined as a tuple $G = (V, E)$, where $V = \{v_0, v_1, \ldots, v_n\}$ is the set of n nodes/vertices and $E = \{e_0, e_1, \ldots, e_m\} \subseteq V \times V$ is the set of m edges/links of G, where an edge connects two vertices.

A graph can be *directed* or *undirected*. In a directed graph, an edge $e_k = (v_i, v_j)$ has a direction with v_i being the starting vertex and v_j the ending vertex. Graphs can be represented by their adjacency, degree, and Laplacian matrices, which are defined as follows:

Definition 2. The *adjacency matrix* A of a graph G with n vertices is an $n \times n$ matrix, where an element a_{ij} in the matrix equals to 1 if there is an edge between node pair v_i and v_j and is 0 otherwise. An adjacency matrix can be *weighted* in which the value of an element represents the weight (such as importance) of the edge it represents.

Definition 3. The *degree matrix* D of a graph G with n vertices is an $n \times n$ diagonal matrix, where an element d_{ii} is the degree of node v_i for $i = \{1, \ldots, n\}$ and all other $d_{ij} = 0$. In undirected graphs, where edges have no direction, the degree of a node refers to the number of edges attached to that node. For directed graphs, the degree of a node can be the number of incoming or outgoing edges of that node, resulting in an *in-degree* or *out-degree* matrix, respectively.

Definition 4. The Laplacian matrix L of a graph G with n vertices is an $n \times n$ matrix, defined as $L = D - A$, where D and A are G's degree and adjacency matrix, respectively.

2.1 Graph Embedding

In order to use graphs in downstream machine learning and data mining applications, graphs and their entities, such as nodes and edges, need to be represented using numerical features. One way to represent a graph is its adjacency matrix. However, an adjacency matrix is memory-consuming for representing very large graphs because its size is $|V| \times |V|$. We can represent a graph and its elements using their features. Specifically, a node in the graph can be represented with a set of features that could help the performance of the representation in a particular application. For example, in an anomaly detection application, the nodes with the densest neighborhood have the potential to be anomalous. Therefore, if we include the in-degree and out-degree of nodes in the node representation, we can more likely detect the anomalous nodes with high accuracy because the anomalous

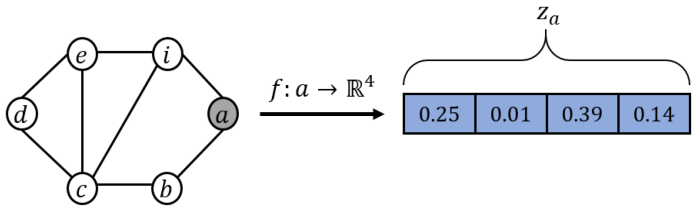

Fig. 1. The graph on the left-hand side consists of 6 nodes $\{a, b, c, d, e, i\}$ and 8 edges. Graph embedding methods map each node of the graph into an embedding vector with dimension d. For demonstration purposes, the node a is embedded into an embedding vector z_a of dimension 4 with given values.

nodes often have larger degrees. However, it could be hard to find features that are important in different applications and can also represent the entire structure of the graph. In addition, it is time-consuming to extract these features manually. Therefore, the graph embedding methods have been proposed, which study the issue of automatically generating representation vectors for the graphs. These methods formulate the graph representation learning as a machine learning task and generate embedding vectors leveraging the structure and properties of the graph as input data. Graph embedding techniques include node, edge, and subgraph embedding techniques, which are defined as follows.

Definition 5. (Node embedding). Let $G = (V, E)$ be a graph, where V and E are the set of nodes and the set of edges of the graph, respectively. Node embedding learns a mapping function $f : v_i \rightarrow \mathbb{R}^d$ that encodes each graph's node v_i into a low-dimensional vector of dimension d such that $d << |V|$ and the similarities between nodes in the graph are preserved in the embedding space.

Figure 1 shows a sample graph and that an embedding method maps node a in the graph to a vector of dimension 4.

Definition 6. (Edge embedding). Let $G = (V, E)$ be a graph, where V and E are the set of nodes and the set of edges of the graph, respectively. Edge embedding converts each edge of G into a low-dimensional vector of dimension d such that $d << |V|$ and the similarities between edges in the graph are preserved in the embedding space.

While edge embeddings can be learned directly from graphs, most commonly they are derived from node embeddings. For example, let $(v_i, v_j) \in E$ be an edge between two nodes v_i and v_j in a graph G and let z_i, z_j be the embedding vectors for nodes v_i, v_j. An embedding vector for the edge (v_i, v_j) can be obtained by applying a binary operation such as hadamard product, mean, weighted-L1 and weighted-L2 on the two node embedding vectors z_i and z_j [93].

Definition 7. (Subgraph embedding). Let $G = (V, E)$ be a graph. Subgraph embedding techniques in machine learning convert a subgraph of G into a low-dimensional vector of dimension d such that $d << |V|$ and the similarities between subgraphs are preserved in the embedding space.

A subgraph embedding vector is usually created by aggregating the embeddings of the nodes in the subgraph using aggregators such as a mean operator.

As node embeddings are the building blocks for edge and subgraph embeddings, almost all the graph embedding techniques developed so far are node embedding techniques. Thus, the embedding techniques we describe in this survey are mostly node embedding techniques unless otherwise stated.

2.2 Graph Embedding Applications

The generated embedding vectors can be utilized in different applications, including node classification, link prediction, and graph classification. Here, we explain some of these applications.

Node Classification. The node classification task assigns a label to the nodes in the test dataset. This task has many applications in different domains. For instance, in social networks, a person's political affiliation can be predicted based on the person's friends in the network. In node classification, each instance in the training dataset is the node embedding vector and the label of the instance is the node label. Different regular classification methods such as Logistic Regression and Random Forests can be trained on the training dataset and generate the node classification scores for the test data. Similarly, graph classification can be performed using graph embedding vectors.

Link Prediction. Link prediction is one of the important applications of node embedding methods. It predicts the likelihood of an edge formation between two nodes. Examples of this task include recommending friends in social networks [255] and finding biological connections in biological networks [1, 53]. For instance, the methods in [1] predict the links between a drug and a target, which could be a disease, a gene, or other drugs in the drug–target interaction network. Link prediction can be formulated as a classification task that assigns a label for edges. Edge label 1 means that an edge is likely to be created between two nodes and the label is 0 otherwise. For the training step, a sample training set is generated using positive and negative samples. Positive samples are the edges the exist in the graph. Negative samples are the edges that do not exist and their representation vector can be generated using the node vectors. Similar to node classification, any classification method can be trained on the training set and predict the edge label for test edge instances.

Anomaly Detection. Anomaly detection is another application of node embedding methods. The goal of anomaly detection is to detect the nodes, edges, or graphs that are anomalous and the time that the anomaly occurs. Anomalous nodes or graphs deviate from normal behavior. For instance, in banks' transaction networks, people who suddenly send or receive large amounts of money or create lots of connections with other people could be potential anomalous nodes. An anomaly detection task can be formulated as a classification task such that each instance in the dataset is the node representation and the instance label is 0 if the node is normal and 1 if the node is anomalous. This formulation necessitates having a dataset with true node labels. One of the issues in anomaly detection is the lack of datasets with true labels. Mitigation offered for this issue in the literature is generating synthetic datasets that model the behaviors of real-world datasets. Another way to formulate the anomaly detection problem, especially in dynamic graphs, is viewing the problem as a change-detection task. In order to detect the changes in the graph, one way is to compute the distance between the graph representation vectors at consecutive times. The time points that the value of this difference is far from the previous normal values, a potential anomaly has occurred [91].

Graph Clustering. In addition to classification tasks, graph embeddings can be used in clustering tasks. This task can be useful in domains such as social networks for detecting communities [148, 154] and biological networks to identify similar groups of proteins [19, 335]. Groups of similar graphs/node/edges can be detected by applying clustering methods such as the K-means method [203] on the graph/node/edge embedding vectors.

Visualization. One of the applications of node embedding methods is graph visualization because node embedding methods map nodes in lower dimensions and the nodes, edges, communities, and different properties of graphs can be seen better in the embedding space. Therefore, graph visualization is very helpful for the research community to gain insight into graph data, especially very large graphs that are hard to visualize.

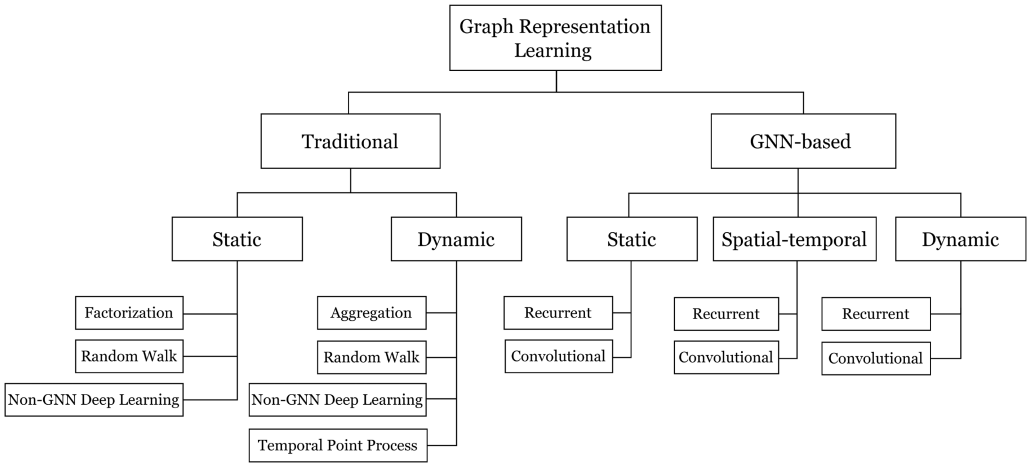

Fig. 2. Categories of graph representation learning methods for non-heterogeneity-aware graphs.

3 TRADITIONAL GRAPH EMBEDDING

The first category of graph embedding methods are traditional graph embedding methods. These methods map the nodes into the lower dimensions using different approaches, such as random walks, factorization methods, and temporal point processes. We review these methods in static and dynamic settings in this section. Figure 2 shows the categories of static and dynamic traditional embedding methods. The upper part of Table 1 lists all the methods that we survey in this category.

3.1 Traditional Static Graph Embedding

The traditional static graph embedding methods have been developed for static graphs. The static graphs do not change over time and have a fixed set of nodes and edges. Graph embedding methods preserve different properties of nodes and edges in graphs, such as node proximities. Here, we define first-order and second-order proximities. Higher orders of proximities can be similarly defined.

Definition 8. (First-order proximity). Nodes that are connected with an edge have first-order proximity. Edge weights are the first-order proximity measures between nodes. Higher weights for edges show more similarity between two nodes connected by the edges.

Definition 9. (Second-order proximity). The second-order proximity between two nodes is the similarity between their neighborhood structures. Nodes sharing more neighbors are assumed to be more similar.

The traditional static graph embedding methods can be put into three categories: *factorization-based*, *random walk–based* and *non-GNN–based deep learning* methods [90, 101]. Below, we review these methods and the techniques they use.

3.1.1 Factorization based. Matrix factorization methods are the early works in graph representation learning. These methods can be summarized in two steps [308]. In the first step, a proximity-based matrix is constructed for the graph where each element of the matrix denoted as P_{ij} is a proximity measure between two nodes i, j. Then, a dimension reduction technique is applied in the matrix to generate the node embeddings in the second step. In the Graph Factorization algorithm [3], the adjacency matrix is used as the proximity measure and the general form of the

Table 1. Graph Embedding Methods in Both Traditional and GNN-Based Categories

Type	Graph	Methods
Trad	Static	Node2vec [93], Deepwalk [232], Graph Factorization [3], GraRep [30], HOPE [222], STRAP [322], HARP [39], LINE [260], SDNE [269], DNGR [31], VGAE [141], AWE [127], PRUNE [147], E[D] [2], ULGE [216], APP [366], CDE [163], GNE [67], DNE [250], DANE [81], RandNE [354], SANE [271], BANE [311], LANE [123], VERSE [264], ANECP [120], NOBE [130], AANE [122], Reinforce2vec [297], REFINE [375], M-NMF [279], struct2vec [238], SNEQ [106], PAWINE [284], FastRP [40], SNS [197], InfiniteWalk [34], EFD [35], NetMF [234], Lemane [353], AROPE [356], NetSMF [233], SPLITTER [73], Ddgk [5], GVNR [26], LouvainNE [21], HONE [242], CAN [208], Methods in [124, 180, 308]
	Dynamic	CTDNE [215], DynNode2vec [204], LSTM-Node2vec [139], EvoNRL [109], DynGEM [91], Dyn-VGAE [205], DynGraph2vec [89], HTNE [383], DynamicTriad [373], DyRep [263], MTNE [117], DNE [69], Toffee [198], HNIP [235], tdGraphEmbed [17], DRLAN [185], TIMERS [355], M2DNE [191], DANE [153], TVRC [249], tNodeEmbed [251], NetWalk [333], DynamicNet [377], Method in [318]
GNN	Static	RecGNN [246], GGNN [165], IGNN [94], Spectral Network [28], GCN [142], GraphSAGE [99], DGN [16], ElasticGNN [179], SGC [290], GAT [266], MAGNA [270], MPNN [86], GN block [15], GNN-FiLM [27], GRNF [336], EGNN [245], BGNN [374], MuchGNN [372], TinyGNN [307], GIN [303], RP-GNN [213], k-GNN [212], PPGN [206], Ring-GNN [50], F-GNN [10], DEGNN [157], GNNML [12], rGIN [244], DropGNN [225], PEG [272], GraphSNN [288], NGNN [346], ID-GNN [326], CLIP [60], APPNP [143], JKNET [304], GCN-PN [361], DropEdge [240], DGN-GNN [371], GRAND [77], GCNII [46], GDC [103], PDE-GCN [72], SHADOW-SAGE [337], ClusterGCN [51], FastGCN [43], LADIES [382], GraphSAINT [338], VR-GCN [44], GBP [45], RevGNN [152], VQ-GNN [63], BNS [317], GLT [47], H2GCN [378], GPR-GNN [52], WRGNN [259], DMP [312], CPGNN [376], U-GCN [134], NLGNN [177], GPNN [314], HOG-GCN [277], Polar-GNN [76], GBK-GNN [68], Geom-GCN [228], GSN [24], MPSN [22], GraphSTONE [188], DeepLPR [49], GSKN [189], SUBGNN [8], DIFFPOOL [324], PATCHY-SAN [217], SEAL [344], DGCNN [345], AGCN [160], DGCN [381], CFANE [224], AdaGNN [66], MCN [149], Method in [159]
	Spatial-temporal	GCRN [248], Graph WaveNet [294], SFTGNN [156], CoST-Net [319], DSTN [223], LightNet [83], DSAN [169], H-STGCN [55], DMSTGCN [102], PredRNN [283], Conv-TT-LSTM [256], ST-ResNet [343], STDN [316], ASTGCN [97], DGCNN [62], DeepETA [291], SA-ConvLSTM [171], STSGCN [253], FC-GAGA [221], ST-GDN [352], HST-LSTM [145], STGCN [329], PCR [310], GSTNet [74], STAR [299], ST-GRU [172], Tssrgcn [48], Test-GCN [6], ASTCN [347], STP-UDGAT [168], STAG-GCN [190], ST-GRAT [227], ST-CGA [351], STC-GNN [287], STEF-Net [166], FGST [321], PDSTN [209], STAN [194], GraphSleepNet [129], DCRNN [164], CausalGNN [274], SLCNN [348], MRes-RGNN [36], Method in [281]
	Dynamic	DyGNN [199], EvolveGCN [226], TGAT [300], CAW-N [282], DySAT [243], EHNA [119], TGN [241], MTSN [184], SDG [79], VGRNN [98], MNCI [176], FeatureNorm [313]

Trad and GNN stand for Traditional and GNN-based graph embedding.

optimization function is as follows:

$$\min_{z_i, z_j} \sum_{v_i, v_j \in V} \left| z_i^T z_j - a_{ij} \right|, \tag{1}$$

where z_i and z_j are the node representation vectors for node v_i and v_j and a_{ij} is the element in the adjacency matrix corresponding to nodes v_i and v_j. In GraRep [30] and HOPE [222], the value of a_{ij} is replaced with other measures of similarity, including higher orders of adjacency matrix, Katz index [135], Rooted page rank [254], and the number of common neighbors. STRAP [322] employs the personalized page rank as the proximity measure and approximates the pairwise proximity measures between nodes to lower the computation cost. In [308], a network embedding update algorithm is introduced to approximately compute the higher-order proximities between node pairs. In [353], it is suggested that using the same proximity matrix for learning node representations may limit the representation power of the matrix factorization–based methods. Therefore, it generates node representations in a framework that learns the proximity measures and SVD decomposition parameters in an end-to-end fashion. Methods in [26, 73, 120, 122, 123, 130, 147, 163, 180, 216, 233, 234, 242, 250, 279, 308, 311, 353, 354, 356, 375] are other examples of factorization-based methods.

3.1.2 Random Walk based. Random walk–based methods have attracted a lot of attention because of their success in graph representation. The main concept that these methods utilize is generating random walks for each node in the graph to capture the structure of the graph and output similar node embedding vectors for nodes that occur in the same random walks. Using co-occurrence in a random walk as a measure of similarity of nodes is more flexible than fixed proximity measures in earlier works and showed promising performance in different applications.

Definition 10. (Random walk). In a graph $G = (V, E)$, a random walk is a sequence of nodes $v_0, v_1, \ldots, v_k$ that starts from node v_0. $(v_i, v_{i+1}) \in E$ and $k + 1$ is the length of the walk. The next node in the sequence is selected based on a probabilistic distribution.

DeepWalk [232] and Node2vec [93] are based on the Word2vec embedding method [211] in **natural language processing (NLP)**. Word2vec is based on the observation that words that co-occur in the same sentence many times have a similar meaning. Node2vec and DeepWalk extend this assumption for graphs by considering that nodes that co-occur in random walks are similar. Therefore, these methods generate similar node embedding vectors for neighbor nodes. The algorithm of these two methods consists of two parts. In the first part, a set of random walks are generated, and in the second part, the random walks are used in the training of a SkipGram model to generate the embedding vectors. The difference between DeepWalk and Node2vec is in the way that they generate random walks. DeepWalk selects the next node in the random walk uniformly from the neighbor nodes of the previous node. Node2vec applies a more effective approach to generating random walks. In this section, we first explain the Node2vec random walk generation and then the SkipGram.

(1) *Random Walk Generation.* Assume that we want to generate a random walk $v_0, v_1, \ldots, v_k$, where $v_i \in V$. Given that the edge (v_{i-1}, v_i) is already passed, the next node v_{i+1} in the walk is selected based on the following probability:

$$P(v_{i+1}|v_i) = \begin{cases} \frac{\alpha_{v_i v_{i+1}}}{Z} & \text{if } (v_{i+1}, v_i) \in E \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

where Z is a normalization factor and $\alpha_{v_i v_{i+1}}$ is defined as

$$\alpha_{v_i v_{i+1}} = \begin{cases} 1/p & \text{if } d_{v_{i-1} v_{i+1}} = 0 \\ 1 & \text{if } d_{v_{i-1} v_{i+1}} = 1 \\ 1/q & \text{if } d_{v_{i-1} v_{i+1}} = 2 \end{cases}, \tag{3}$$

where $d_{v_{i-1} v_{i+1}}$ is the length of the shortest path between nodes v_{i-1} and v_{i+1} and takes values from $\{0, 1, 2\}$. The parameters p and q guide the direction of the random walk and can be set by the user. A large value for parameter p encourages global exploration of the graph and avoids returning to the nodes that are already visited. A large value for q, on the other hand, biases the walk toward local exploration. With the use of these parameters, Node2vec creates a random walk that is a combination of **breadth-first search (BFS)** and **depth-first search (DFS)**.

(2) *SkipGram.* After generating random walks, the walks are input to a SkipGram model to generate the node embeddings. SkipGram learns a language model, which maximizes the probability of sequences of words that exist in the training corpus. The objective function of SkipGram for node representation is

$$\max_{\Phi} \sum_{v_i \in V} \log P(N(v_i) | \Phi(v_i)), \tag{4}$$

where $N(v_i)$ is the set of neighbors of node v_i generated from the random walks. Assuming independency among the neighbor nodes, we have that

$$P(N(v_i) | \Phi(v_i)) = \prod_{v_k \in N(v_i)} P(\Phi(v_k) | \Phi(v_i)). \tag{5}$$

The conditional probability of $P(\Phi(v_k)|\Phi(v_i))$ is modeled using a softmax function:

$$P(\Phi(v_k) | \Phi(v_i)) = \frac{\exp(\Phi(v_k)\Phi(v_i))}{\sum_{v_j \in V} \exp(\Phi(v_j)\Phi(v_i))}. \tag{6}$$

The softmax function nominator is the dot product of the node representation vectors. Since the dot product between two vectors measures their similarity, by maximizing the softmax function for neighbor nodes, the generated node representations for neighbor nodes tend to be similar. Computing the denominator of the conditional probability is time-consuming between the target node and all the nodes in the graph. Therefore, DeepWalk and Node2vec approximate it using hierarchical softmax and negative sampling, respectively.

In HARP [39], a graph-coarsening algorithm is introduced that generates a hierarchy of smaller graphs as $G_0, G_1, \ldots, G_L$ such that $G_0 = G$. Starting from the G_L, which is the smallest graph, the node embeddings that are generated for G_i are used as initial values for nodes in G_{i-1}. This method avoids getting stuck in the local minimum for DeepWalk and Node2vec because it initializes the node embeddings with better values in the training process. The embedding at each step can be created using DeepWalk [232] and Node2vec [93] methods. LINE [260] is not based on random walks. However, because it is computationally related to DeepWalk and Node2vec, its results are usually compared with them. LINE generates node embeddings that preserve the first-order and second-order proximities in the graph using a loss function that consists of two parts. In the first part, L_1, it minimizes the reverse of the dot product between connected nodes. In the second part, L_2, for preserving the second-order proximity, it assumes that nodes that have many connections in common are similar. LINE trains two models that minimize L_1 and L_2 separately; then, the embedding of a node is the concatenation of its embeddings from two models. Methods in [21, 34, 35, 124, 127, 197, 238, 271, 284, 297, 366] are some other variants of random walk–based methods.

3.1.3 Non-GNN-based Deep Learning. SDNE [269] is based on an autoencoder that tries to reconstruct the adjacency matrix of a graph and captures nodes' first-order and second-order proximities. To that end, SDNE jointly optimizes a loss function that consists of two parts. The first part preserves the second-order proximity of the nodes and minimizes the following loss function:

$$L_1 = \sum_{v_i \in V} |(x_i - x_i') \odot b_i|, \tag{7}$$

where x_i is the row corresponding to node v_i in the graph adjacency matrix and x_i' is the reconstruction of x_i. b_i is a vector consisting of b_{ij}s for j from 1 to n (the number of nodes in the graph). If $a_{ij} = 0, b_{ij} = 1$; otherwise, $b_{ij} = \beta > 1$. a_{ij} is the element corresponding to nodes v_i and v_j in the adjacency matrix. Using b_i, SDNE assigns more penalty for the error in the reconstruction of the non-zero elements in the adjacency matrix to avoid reconstructing only zero elements in sparse graphs. The second part captures the first-order similarity and optimizes $L2$:

$$L_2 = \sum_{(v_i, v_j) \in E} a_{ij} |(z_i - z_j)|, \tag{8}$$

where z_i and z_j are the embedding vectors for nodes v_i and v_j, respectively. In this way, a higher penalty is assigned if the difference between the embedding vectors of two nodes connected by an edge is higher, resulting in similar embedding vectors for nodes connecting with an edge. This loss is based on ideas from Laplacian Eigenmaps [18]. SDNE jointly optimizes L_1 and L_2 to generate the node embedding vectors. The embedding method DNGR [31] is also very similar to SDNE with the difference that DNGR uses pointwise mutual information of two nodes co-occurring in random walks instead of the adjacency matrix values. VGAE [141] is a variant of variational autoencoders [140] on graph data. The variational graph encoder encodes the observed graph data, including the adjacency matrix and node attributes into low-dimensional latent variables.

$$q(Z|A, X) = \prod_{i=1}^{N} q(z_i|A, X),$$
$$with\ q(z_i|A, X) = N\left(z_i|\mu_i, diag\left(\sigma_i^2\right)\right)$$

where z_i is the embedding vector for node v_i, μ_i is a mean vector and σ_i is the log standard deviation vector of node v_i. A and X are the adjacency matrix and attribute matrix of the graph, respectively. The variational graph decoder decodes the latent variables into the distribution of the observed graph data as follows:

$$p(A|Z) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(a_{i,j}|z_i, z_j),$$
$$with\ p(a_{i,j} = 1|z_i, z_j) = sigmoid\left(z_i^T, z_j\right).$$

The model generates embedding vectors that minimize the distance between the p and q probability distributions using the KL-divergence measure, SGD, and reparametrization trick. Other works in [2, 5, 40, 67, 81, 106, 208, 264] also learn node embeddings using non-GNN-based deep learning models.

3.2 Traditional Dynamic Graph Embedding

Most real-world graphs are dynamic and evolve, with nodes and edges added and deleted from them. Dynamic graphs are represented in two ways in the dynamic graph embedding studies: discrete-time and continuous-time.

Definition 11. (Discrete-time dynamic graphs). In discrete-time dynamic graph modeling, dynamic graphs are considered a sequence of graphs' snapshots at consecutive time points. Formally, dynamic graphs are represented as $G = G_0, G_1, \ldots, G_T$, in which G_i is a snapshot of the graph G at timestamp i. The dynamic graph is divided into graph snapshots using time granularity such as hours, days, months, and years depending on the dataset and applications.

Definition 12. (Continuous-time dynamic graphs). In continuous-time dynamic graph modeling, the time is continuous and the dynamic graph can be represented as a sequence of edges over time. The dynamic graph can also be modeled as a sequence of events in which events are the changes in the dynamic graphs, such as adding/deleting edges/nodes.

Definition 13. (Dynamic graph embedding). We can use either the discrete-time or the continuous-time approach for representing a dynamic graph. Let $G_t = (V_t, E_t)$ be the graph at time t, with V_t, E_t as the nodes and edges of the graph. Dynamic graph embedding methods map nodes in the graph to a lower-dimensional space d such that $d << |V|$.

The naïve way to generate dynamic graph embeddings is to apply static graph embedding methods on dynamic graphs. Static methods can be applied to discrete-time and continuous-time dynamic graphs, but the generated embeddings may not fully represent dynamic graphs. In discrete-time graphs, a static graph embedding method can be applied to each graph snapshot separately. However, the generated embeddings at different snapshots are at different embedding spaces. Therefore, the embeddings generated at different time points are not comparable and the embeddings do not capture the evolution of the graph over time. Similarly, in the continuous-time dynamic graphs, the static graph embedding methods do not consider the order in which each interaction occurs and lose the time information. In addition, static methods must be rerun from scratch to learn the new embedding with the arrival of new interactions, which is very time-consuming. Due to these issues, different dynamic graph embedding methods have been proposed for dynamic graph representation learning. Here, we provide an overview of the dynamic embedding methods and put these methods into four categories: Aggregation based, Random walk based, Non-GNN based deep learning and Temporal point process based [14, 136].

3.2.1 Aggregation based. Aggregation-based dynamic graph embedding methods aggregate the dynamic information of graphs to generate embeddings for dynamic graphs. These methods can fall into two groups:

(1) Aggregating the temporal features. In these methods, the evolution of the graph is simply collapsed into a single graph and the static graph embedding methods are applied on the single graph to generate the embeddings. For example, the aggregation of the graph over time could be the sum of the adjacency matrices for discrete-time dynamic graphs [167] or the weighted sum, which gives more weights to recent graphs [249]. One drawback of these methods is that they lose the time information of graphs that reveals the dynamics of graphs over time. For instance, there is no information about when any edge was created. *Factorization-based models* can also fit into the aggregation-based category. The reason is that factorization-based models save the sequence of graphs over time in a three-dimensional tensor $\in \mathbb{R}^{|V|\times|V|\times T}$ (T is a time dimension) and then apply factorization on this tensor to generate the dynamic graph embeddings [70, 153, 185, 355].

(2) Aggregating the static embeddings. These aggregation methods first apply static embedding methods on each graph snapshot in the dynamic graph sequence. Then, these embeddings are aggregated into a single embedding matrix for all the nodes in the graph. These methods usually aggregate the node embeddings by considering a decay factor that assigns a lower weight to older graphs [318, 377]. In another type of these methods, the sequence of graphs from time 0 to $t-1$

are fit into a time-series model such as ARIMA, which predicts the embedding of the next graph at time $t + 1$ [54].

3.2.2 Random Walk based. Random walk–based approaches extend the concept of random walks in the static graphs for dynamic graphs. Random walks in dynamic graphs capture the time dependencies between graphs over time in addition to the topological structure of each of the graph snapshots. Depending on the definition of random walks, different methods include the temporal information of the graphs differently. CTDNE [215] defines a temporal walk to capture time dependencies between nodes in dynamic graphs. CTDNE considers a continuous-time dynamic graph such as graph $G = (V, E_T, T)$, where V, E_T, T are nodes and edges of the graph and time $T : E \rightarrow \mathbb{R}^+$. Each edge e in this graph is represented by a tuple (u, v, t), where u, v are the nodes connected by the edge and t is the time of occurrence of that edge.

Definition 14. (Temporal walk). A temporal walk is a sequence of nodes $v_0, v_1, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E_T$ and $t_{(v_{i-1}, v_i)} \leq t_{(v_i, v_{i+1})}$.

An important concept in CTDNE is that time is respected in selecting the next edge in a temporal walk. In order to generate these temporal walks, first a time and a particular edge in that time, $e = (u, v, t_e)$, is selected based on one of three probability distributions: uniform, exponential, and linear. The uniform probability for an edge e is $p(e) = 1/|E_T|$. The exponential probability is

$$p(e) = \frac{exp(t_e - t_{min})}{\sum_{e' \in E_T} exp(t'_e - t_{min})}, \tag{9}$$

where t_{min} is the minimum time of an edge in the graph. Using exponential probability distribution, edges that appear at a later time are more likely to be selected. After selecting $e = (u, v, t_e)$, the next node in the temporal walk is selected from the neighbors of node v in time $t_e + k$, where $k > 0$ again using one of the uniform, exponential, or linear probability distributions. The generated temporal walks are then input to a SkipGram model and the temporal node representation vectors are generated.

DynNode2vec [204] is a dynamic version of Node2vec [93]. It uses a discrete-time approach for dynamic graph representation learning. This method represents the dynamic graph as a sequence of graph snapshots over time as $G_0, G_1, \ldots, G_T$. The embedding for the graph at time 0, G_0, is computed by applying Node2vec on G_0. Then, for next time points, the SkipGram model of G_{t+1} is initialized using node representations from G_t for nodes that are common between consecutive time points. New nodes will be initialized randomly. In addition, DynNode2vec does not generate random walks at each timestep i from scratch. Instead, it uses random walks from the previous time $i-1$ and only updates the ones that need to be updated. This method has two advantages. First, it saves time because it does not generate all the walks in each step. Second, since the SkipGram model at time t is initialized with weights from time $t-1$, embedding vectors of consecutive times are in the same embedding space, embedding vectors of nodes change smoothly over time. and the model converges faster. LSTM-Node2vec [139] captures both the static structure and evolving patterns in graphs using a **long short-term memory (LSTM)** autoencoder and a Node2vec model. The dynamic graph is represented as a sequence of snapshots over time as $G_0, G_1, \ldots, G_T$. For each graph G_i at time t_i, first, a set of temporal walks is generated for each node in the graph. Each temporal random walk of a node v is represented as $w_0, w_1, \ldots, w_L$ of length L, where w_j is a neighbor of the node v at time t_j in graph G_j and $t_j < t_{j+1}$. These temporal walks demonstrate changes in the neighborhood structure of the node before time t_i. EvoNRL [109] focuses on maintaining a set of valid random walks for the graph at each time point so that the generated node embeddings using these random walks stay accurate. To that end, EvoNRL updates the existing

random walks from previous time points instead of generating random walks from scratch. Specifically, it considers four cases of edge addition, edge deletion, node addition and node deletion for evolving graphs and updates the affected random walks accordingly. For instance, in the edge addition case, EvoNRL finds random walks containing the nodes that are connected by the updated link and updates those walks. However, updating random walks is time-consuming, especially for large graphs. Therefore, EvoNRL proposed an indexing mechanism for fast retrieval of random walks. Other examples of this category include [17, 69, 198, 333].

3.2.3 Non-GNN-based Deep Learning. This type of dynamic graph embedding method uses deep learning models such as **recurrent neural networks (RNNs)** and autoencoders. DynGEM [91] is based on the static deep learning–based graph embedding method SDNE [269]. Let the dynamic graph be a sequence of graph snapshots $G_0, G_1, \ldots, G_t$. The embeddings for graph G_0 are computed using a SDNE model. The embedding of G_i is obtained by running an SDNE model on G_i that is initialized with the embeddings from G_{i-1}. This initialization leads to generating node embeddings at consecutive time points that are in the same embedding space and can reflect the changes in the graph at consecutive times accurately. As the size of the graph can change over time, DynGEM uses Net2WiderNet and Net2DeeperNet to account for bigger graphs [37]. Dyn-VGAE [205] is a dynamic version of VGAE [141]. The input to dyn-VGAE is the dynamic graph as a sequence of graph snapshots, $G_0, G_1, \ldots, G_T$. At each time point, the embedding of the graph snapshot G_i is obtained using VGAE. However, the loss of the model at time t has two parts. The first part is related to VGAE loss and the second loss is a KL divergence measure that minimizes the difference between two distributions as follows:

$$L_s^t = KL[q_t(Z_t|X_t, A_t)||N(Z_{t-1}, \sigma^2)], \tag{10}$$

where $q_t(Z_t|X_t, A_t)$ is the distribution of latent vectors at time t and $N(Z_{t-1}, \sigma^2)$ is a normal distribution with mean Z_{t-1} and standard deviation σ. This loss places the current latent vectors Z_t near latent vectors of previous time point Z_{t-1}. The loss function of all the models for the graph at time 0 to T are jointly trained. Therefore, the generated representation vectors preserve both the structure of the graph at each time point and evolutionary patterns obtained from previous time points. Dyngraph2vec [89] generates embeddings at time t using an autoencoder. This method inputs adjacency matrices of previous times $A_0, A_1, \ldots, A_{t-1}$ to the encoder and using the decoder reconstructs the input and generates the embeddings at time t. Dyngraph2vec proposes several variants using a fully connected model or an RNN/LSTM model for the encoder and the decoder: dyngraph2vecAE, dyngraph2vecAERNN and dyngraph2vecRNN. Dyngraph2vecAE uses an autoencoder, dyngraph2vecAERNN is based on an LSTM autoencoder, and dyngraph2vecAERNN has an LSTM enocoder and a fully connected decoder. Other examples of non-GNN-based deep learning methods include [235, 251].

3.2.4 Temporal Point Process Based. This class of the dynamic graph embedding methods assumes that the interaction between nodes for creating the graph structure is a stochastic process and models it using temporal point processes. HTNE [383] generates embeddings for dynamic graphs by modeling the neighborhood formation of nodes as a Hawkes process. In a Hawkes process modeling, the occurrence of an event at time t is influenced by events that occur before time t and a conditional intensity function characterizes this concept. Let $G = (V, E, A)$ be the temporal network, where V, E, A are the nodes, edges, and events. Each edge (v_i, v_j) in this graph is associated with a set of events $a_{ij} = \{a_1 \rightarrow a_2 \rightarrow \ldots\} \subset A$, where each a_i is an event at time i.

Definition 15. (Neighborhood Formation Sequence). A neighborhood formation sequence for a node v_i is a series of neighborhood arrival events $\{v_i : (u_0, t_0) \rightarrow (u_1, t_1) \ldots \rightarrow (u_k, t_k)\}$, where u_i is a neighbor of v_i that occurs at time t_i.

HTNE models the neighborhood formation for a node v using the neighborhood formation sequence H_v. The probability that an edge forms between node v and a target neighbor u at time t is represented using the following formula:

$$p(u|v, H_v) = \frac{\lambda_{u|v}(t)}{\sum_{u'} \lambda_{u'|v}(t)}, \tag{11}$$

where $\lambda_{y|x}(t)$ is defined as

$$\lambda_{u|v}(t) = exp\left(\mu_{u,v} + \sum_{h,u} \alpha_{h,u} \kappa(t - t_h)\right). \tag{12}$$

$\lambda_{u|v}(t)$ is the conditional intensity function of a Hawkes process, which is the arrival rate of target neighbor u for node v at time t given the previous neighborhood formation sequence. $\mu_{u,v}$ is a base rate of edge formation between u, v and it is equal to $|z_u - z_v|$. h is a historical neighbor of the node v in the neighborhood formation sequence in a time before t. $\alpha_{h,u}$ is the degree that the historical neighbor h is important for u and it equals $|z_h - z_u|$. $\kappa(t - t_h)$ is a decay factor to control the intensity of influence of a historical node on u. HTNE generates embedding vectors that maximize the $\sum_{v \in V} \sum_{u \in H_v} p(u|v, H_v)$ for all the nodes using SGD and negative sampling to deal with a large number of computations in the denominator of the probability function. DyRep [263] captures the dynamic of graphs using two temporal point process models. DyRep argues that in the evolution of a graph, two types of events occur: communication and association. Communication events are related to node interactions and association events are the topological evolution and these events occur at different rates. For instance, in a social network, a communication event such as liking a post from someone happens much more frequently than an association event such as creating a new friendship. DyRep represents these two events as two temporal point process models. MTNE [117] is based on the concepts of triad motif evolution and the Hawkes process. This method considers the evolution of graphs as the evolution of motifs in the graphs and models that evolve using a Hawkes process. MTNE argues that a model such as HTNE based on neighborhood formation processes considers network evolution at edge and node levels and cannot reflect network evolution very well. Therefore, MTNE models dynamics in a graph as a subgraph (motif) evolution process. M2DNE [191] is another example of temporal point process–based dynamic embedding methods.

3.2.5 Other Methods. DynamicTriad [373] generates dynamic graph embeddings by modeling the triad closure process, which is a fundamental process in the evolution of graphs.

Definition 16. (Triad closure process). Let (v_i, v_j, v_k) be an open triad in the graph at time t, which means that there are two edges (v_i, v_j) and (v_j, v_k) in the graph but no edge exists between v_i and v_k. It is likely that an edge forms between v_i and v_k at time $t + 1$ because of the influence of node v_j and closes the open triad.

DynamicTriad computes the probability that an open triad (v_i, v_j, v_k) evolves into a closed triad under the influence of v_j at time t as $p^t_{tr}(i, j, k)$. An open triad can evolve in two ways: (1) It becomes closed because of the influence of any one of the neighbors or (2) stays open because no neighbor could influence the creation of the open link. These two evolution traces are reflected in DynamicTriad loss function by maximizing $(p^t_{tr}(i, j, k))^{\alpha_{ijk}} \times (1 - p^t_{tr}(i, j, k))^{(1-\alpha_{ijk})}$ for open triad samples that close under the influence of a neighbor and $1 - p^t_{tr}(i, j, k)$ for those samples that do not close. $\alpha_{ijk} = 1$ if an open triad closes at time $t + 1$. The loss function also utilizes social homophily and temporal smoothness regularizations. *Social homophily smoothness* assumes that nodes that are highly connected are more similar and should have similar embeddings. *Temporal smoothness*

Table 2. Comparison of Traditional and GNN-Based Graph Representation Learning

Category	Advantages	Disadvantages
Traditional	Higher expressive power, scalable in some categories	Not generalizable to unseen nodes, not considering node/edge attributes easily
GNN-based	Generalize to unseen nodes, consider node/edge attributes, can do both task-specific and node similarity–based training	Expressive power, scalability, over-smoothing, over-squashing, homophily assumption and catastrophic forgetting. (More details in Section 4.6)

assumes that the network evolves smoothly and, therefore, the distance between embeddings of a node at consecutive times should be small.

3.2.6 Time Efficiency of Dynamic Graph Embeddings. The time efficiency of dynamic graph embedding methods depends on the techniques they use for generating graph embeddings and the requirements of the dynamic environment. For instance, dynnode2vec, a dynamic graph embedding method, is faster than the static Node2vec method in generating graph embeddings of a series of graph snapshots over time. The reason is twofold. First, dynnode2vec initializes the SkipGram model at each timestep with the weights from the previous time point, which leads to faster model convergence compared with random initialization in Node2vec. Second, this method only generates random walks for the changed nodes at each timestamp, which is faster than generating random walks for all the nodes in Node2vec [204]. However, Node2vec is faster than the dynamic method LSTM-Node2vec, as LSTM-Node2vec trains an LSTM autoencoder model at each time point, which is time-consuming. LSTM-Node2vec, in turn, is faster than two other dynamic methods, dyngraph2vecAE and dyngraph2vecAERNN, as these two models reconstruct adjacency matrices of the graphs over time compared with temporal random walks in LSTM-Node2vec [139]. The requirements of the dynamic settings are also important in the training speed of models. For example, the static methods are offline and can scan the data multiple times. However, some dynamic methods need to be online so that they scan the data only once and incrementally update the model; therefore, they are faster than using a static method to learn embeddings at each time point.

4 GNN-BASED GRAPH EMBEDDING

Graph Neural Network (GNN)–based graph embedding methods are the second category of graph embedding methods, which employ GNNs to generate embeddings. These methods are different from traditional methods in that the GNN-based methods generalize well to unseen nodes. In addition, they can better take advantage of node/edge attributes. Table 2 shows the advantages and disadvantages of the different categories of graph embedding methods. In this section, we first introduce GNNs. Then, three categories of GNN-based methods —static, spatial-temporal, and dynamic GNNs (see Figure 2 for subcategories and Table 1 for the list of methods in each category) — and their real-world applications are surveyed. Finally, we summarize the limitations of GNNs and the proposed solution to these limitations.

4.1 Introduction to GNNs

A GNN is a deep learning model that generates a node embedding by aggregating the node's neighbors' embeddings. The GNN's intuition is that a node's state is influenced by its interactions with its neighbors in the graph. Below, we explain GNN's basic architecture and training.

4.1.1 Basic Architecture. GNNs can generate node representation vectors by stacking several GNN layers. Let h_i^l represent the node embeddings for node i at layer l. Each GNN layer takes as

input the node embeddings. The node representations for node i at each layer $l+1$ are updated using the following formula:

$$h_i^{(l+1)} = f\left(h_i^l, \sum_{j \in N(i)} g(i,j)\right), \tag{13}$$

where f and g are learnable functions and $N(i)$ are the neighbors of node i. h_i^0 are the node i initial features. At each layer, the embedding of the node i is obtained by aggregating the embeddings of the node's neighbors. After passing through L GNN layers, the final representation of node i is h_i^L, which is the aggregation the node's neighbors of L hops away from the node.

4.1.2 GNN Training. GNNs can be trained in supervised, semi-supervised, and unsupervised frameworks. In supervised and semi-supervised frameworks, different prediction tasks focusing on nodes, edges, and graphs can be employed for training the model. Here, we describe the other layers stacked after GNN layers to generate the prediction results.

- Node-focused: For node-level prediction such as node classification, the GNN layers output node representations. Then, using a **multilayer perceptron (MLP)** or a softmax layer, the prediction output is generated.
- Edge-focused: In edge-focused prediction, including link prediction, given two nodes' representations, a similarity function or an MLP is used for the prediction task.
- Graph-focused: In graph-focused tasks such as graph classification, a graph representation is often generated by applying a readout layer on node representations. The readout function can be a pooling operation that aggregates representations of a graph's nodes to generate the graph representation vector. A clique pooling operation has also been proposed, which aggregates a graph's cliques for generating the graph embedding [195].

A typical way to train a GNN in a node classification task is by applying the cross entropy loss function as follows:

$$L = \sum_{i \in V_{train}} y_i \log(\sigma(h_i^T \theta)) + (1 - y_i)\log(1 - \sigma(h_i^T \theta)), \tag{14}$$

where h_i is the embedding of node u, which is the output of the last layer of GNN and y_u is the true class label of the node and θ are the classification weights. Figure 3 shows a general framework for training a GNN using a node classification task. There are three types of nodes in a node classification in GNNs [100]:

- Training nodes: Nodes whose embeddings are computed in the last layer of GNNs and are included in the loss function computation.
- Transductive test nodes: Nodes whose embeddings are computed in the GNN but are not included in the loss function computation.
- Inductive test nodes: They are not included in the GNN computation and loss function.

Transductive node classification in GNNs is equivalent to semi-supervised node classification. It refers to testing on transductive test nodes that are observed during training but their labels are not used. On the other hand, inductive node classification means that the testing is on inductive test nodes (unseen nodes); these test nodes and all their adjacent edges are removed during training. The loss function for graph classification and link prediction tasks can be similarly defined using graph representations and pairwise node representations. In an unsupervised framework for GNN training, node similarities obtained from co-occurrence of nodes in the graph random walks can be used for model training. GNNs often compute node representations using a graph-level implementation to avoid redundant computations for neighbors that are shared among nodes. In addition,

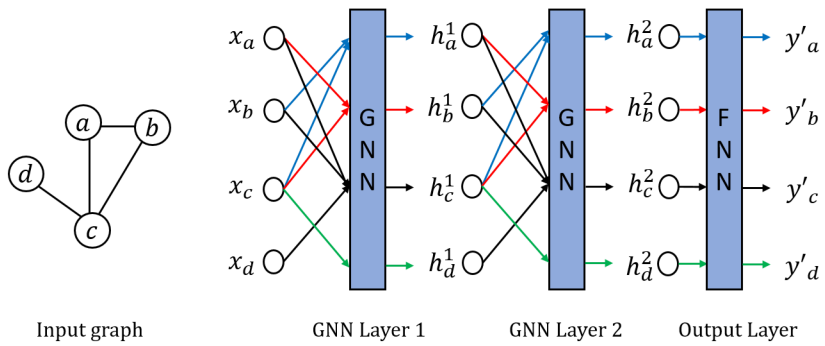

Fig. 3. A general supervised framework for training GNN layers. Two GNN layers are applied on an input graph to compute the node representation vectors for its nodes. The colors on arrows show neighbors of a target node that are aggregated to generate the target node representation. x_a is the feature vector of node a and h_a^1 and h_a^2 are the representation vectors generated for the node a after applying the first and second GNN layers. The generated embeddings are used in a node classification task. y'_a is the predicted label for the node a.

formulating the message passing operations as matrix multiplications is computationally cheap. As an example for a basic GNN, the node embedding computation formula can be reformulated as

$$H^{(l+1)} = \sigma(\hat{A}H^l W^l), \tag{15}$$

where H^l contains the embedding of all the nodes in layer l and W_l is the weight matrix at layer l. $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I_n$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, I_n is an identity matrix, and A, D are the graph's adjacency and degree matrices. The graph-level implementation avoids redundant computations; however, it needs to operate on the whole graph, which may lead to memory limitations. Various methods have been proposed to alleviate the memory complexities of GNNs, which will be discussed in Section 4.6.

4.1.3 Other Important Concepts in GNNs. In this section, we define some of the concepts that are frequently used in the GNN-based graph representation literature.

Receptive Field. The receptive field of a node in GNNs are the nodes that contribute to the final representation of the node. After passing through each layer of the GNN, the receptive field of a node grows one step towards its distant neighbors.

Graph Isomorphism. Two graphs are isomorphic if they have a similar topology. Some of the early works on GNNs, such as GCN [142] and GraphSAGE [99], fail to distinguish non-isomorphic graphs in some cases.

Weisfeiler & Lehman (WL) test. The WL test [150] is a classic algorithm for testing graph isomorphism. It has been shown that the representation power of the message passing GNNs is upper bounded by this test [303]. The WL test successfully determines isomorphism between different graphs, but there are some corner cases in which it fails. Similarly, GNNs fail in those cases. The simple way of thinking about how this test works is that it first counts the number of nodes in two graphs. If two graphs have a different number of nodes, they are different. If two graphs have a similar number of nodes, it checks the number of immediate neighbors of each node. If the number of immediate neighbors of each node is the same, it goes on to check the second-hop neighbors of nodes. If two graphs are similar in all these cases, then they are identical or isomorphic.

Skip connections. A skip connection in deep architectures means skipping some layers in the neural network and feeding one layer's output as an input to the next layers, not just the immediate

next layer. A skip connection helps in alleviating the vanishing gradient effect and preserving information from previous layers. For instance, skip connections are used in the GraphSAGE [99] update step. This method concatenates the node representation at the previous level with the aggregated representation from node neighbors from the previous layer in the update step. This way, it preserves more node-level information in the message passing.

4.2 Static Graph Neural Networks

Static GNN-based graph embedding methods are suitable for graph representation learning on static graphs, which do not change over time. These methods can be divided into two classes: Recurrent GNNs and Convolutional GNNs, which will be explained below.

4.2.1 Recurrent Graph Neural Networks (RecGNNs). RecGNNs are the early works on GNNs that are based on RNNs. The original GNN model proposed by Scarselli et al. [246] used the assumption that nodes in a graph constantly exchange information until they reach an equilibrium. In this method, the representation of node v at iteration t, h_v^t is defined using the following recurrence equation:

$$h_v^t = \sum_{u \in N(v)} f\left(x_v, x^e_{(v,u)}, h_u^{(t-1)}, x_u\right), \tag{16}$$

where f is a recurrent function. $N(v)$ is a set of neighborhood nodes of node v. x_v, x_u are feature vectors of nodes v, u and $x^e_{(v,u)}$ is the feature vector of the edge (u, v). This GNN model recursively runs until convergence to a fixed point. Therefore, the final representation h_v^T in this method is a vector for which $h_v^T = f(h_v^{T-1})$. In this model, h_v^0 is initialized randomly. The initialization of node representation vectors does not matter in this model because the function f recursively converges to the fixed point using any value as an initialization. For learning the model parameters, the states h_v^t are iteratively computed until the iteration T. An approximate fixed point solution is obtained and used in a loss function to compute the gradients. This model has several limitations. One limitation is that if T is large, the iterative computation of node representation until convergence is time-consuming. Furthermore, the node representations obtained from this model are more suitable for graph representation than node representation, as the outputs are very smooth. GGNN [165] uses a **gated recurrent unit (GRU)** as the recurrent function in the original RecGNN method proposed by Scarselli et al. [246]. The advantage of using a GRU is that the number of recurrence steps is fixed and the aggregation does not need to continue until convergence. The h_v^t formula is as follows:

$$h_v^t = GRU\left(h_v^{(t-1)}, \sum_{u \in N(v)} h_u^{(t-1)}\right). \tag{17}$$

The h_v^0 are initialized with node features. The **Implicit Graph Neural Net (IGNN)** [94] is another recurrent GNN that generates node representations by iterating until convergence with no limit on the number of neighbor hops. However, it guarantees the existence of the solution for the equilibrium equations by defining the concept of well-posedness for GNNs, which was previously defined for neural networks [71] and enforcing it at the training time.

4.2.2 Convolutional Graph Neural Networks (ConvGNNs). ConvGNNs are a well-known category of GNNs. These methods generate node embeddings using the concept of convolution in graphs. The difference between ConvGNNs and RecGNNs is that ConvGNNs use **convolutional neural network (CNN)**–based layers to extract node embeddings instead of RNN layers in RecGNNs. There are three key characteristics in CNNs that make them attractive in graph representations. (1) Local connections: CNN can extract local information from neighbors for each

node in the graph; (2) shared weights: weight sharing in node representation generates node embeddings that consider the information of other nodes in the graph; and (3) multiple layers: each layer of convolution can explore a layer of proximities between nodes [369]. ConvGNNs have two categories that can overlap: spectral-based and spatial-based methods. The spectral-based methods have roots in graph signal processing and define graph signal filters. The spatial-based methods are based on information propagation and message-passing concepts from RecGNNs and are more preferred than spectral methods because of efficiency and flexibility. Here, we explain these two categories in more detail.

(1) Spectral based. Spectral-based GNNs utilize mathematical concepts from graph signal processing. **Spectral Network** [28] is one of the early works that defines convolution operations on graphs. Here, we define some of the main concepts shared among spectral-based GNNs.

Definition 17. (Graph Signal). In graph signal processing, a graph signal $x \in \mathbb{R}^n$ is an array of n real or complex values for n nodes in the graph.

Definition 18. (Eigenvectors and eigenvalues (spectrum)). Let $L = D - A$ be the graph Laplacian of graph G, where D, A are the graph's degree matrix and adjacency matrix. The normalized graph Laplacian matrix is $L_N = I_n - D^{-1/2}AD^{-1/2}$, which can be factorized as $L_N = U\Lambda U^T$. U is the matrix of eigenvectors and Λ is the diagonal matrix of ordered eigenvalues. The set of eigenvalues of a matrix are also called the *spectrum of the matrix*.

Definition 19. (Graph Fourier transform). The graph Fourier transform is $F = U^T x$, which maps the graph signal x to a space formed by the eigenvectors of L_N.

Definition 20. (Spectral Graph Convolution). The spectral graph convolution of the graph signal x with a filter $g \in \mathbb{R}^n$ is defined as

$$x * g = F^{-1}(F(x) \odot F(g)) = U(U^T x \odot U^T g) \tag{18}$$

$$\Rightarrow x * g_\theta = U g_\theta U^T x \tag{19}$$

$$\text{where } g_\theta = diag(U^T g). \tag{20}$$

Different spectral-based ConvGNNs use a different graph convolution filter g_θ. For instance, Spectral CNN [28] defines g_θ as a set of learnable parameters. One of the main limitations of this method is the eigenvalue decomposition computational complexity. This limitation was resolved by applying several approximations and simplification in future works. A **Graph Convolutional Network (GCN)** [142] uses a layerwise propagation rule based on multiplying the first-order approximation of localized spectral convolution filter g_θ with a graph signal x as follows:

$$x * g_\theta = \theta(I_n + D^{-1/2}AD^{-1/2})x, \tag{21}$$

where D, A are the degree matrix and adjacency matrix of a graph G and I_n is an identity matrix with 1 on the diagonal and 0 elsewhere. θ represents the filter parameters. The GCN also modifies the convolution operation into a layer defined as $H = X * g_\Theta = f(\bar{A}X\Theta)$, where f is an activation function and $\bar{A} = I_n + D^{-1/2}AD^{-1/2}$. Using a renormalization trick, $\bar{A}$ is replaced with $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I_n$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Therefore, the formulation of h_v^{l+1} for node v becomes:

$$h_v^{l+1} = f\left(\Theta^l\left(\sum_{u \in \{N(v) \cup v\}} \hat{A}_{v,u} x_u\right)\right), \tag{22}$$

where $\bar{A}$ is a constant and is approximately computed in a preprocessing step. $N(v)$ is the set of neighbors of a node v. Therefore, h_v^{l+1} value can be roughly approximated as

$$h_v^{l+1} \approx f\left(\Theta^l.Mean\left(h_v^l \cup \left\{h_u^l, \forall u \in N(v)\right\}\right)\right). \tag{23}$$

In a neural network setting, f is an activation function such as ReLU and Θ^l is the matrix of parameters at layer l. The GCN can be viewed as a spatial-based GNN because it updates the node embeddings by aggregating information from neighbors of nodes. In [159], a spectral-based model is proposed that jointly learns relations between nodes and relations between attributes of nodes. The node embeddings in this model are the output of a 2D spectral graph convolution defined as $Z = GXF$. In this formula, X is a node feature matrix and G and F are an object graph convolutional filter and an attribute graph convolutional filter, respectively. The object graph convolutional filter is defined by designing a filter on the adjacency matrix of the graph. For defining the attribute graph convolutional filter, an attribute affinity graph is constructed on the original graph by applying either positive pointwise mutual information or word embedding–based **K-nearest Neighbor (KNN)** on the attributes of the nodes. **Directional Graph Networks (DGNs)** [16] defines directions for information propagation in the graph using vector fields to improve the message passing in a specific direction in the current GNNs. In this method, the contribution of a neighbor node depends on its alignment with the receiving node's vector field. The vector fields denoted by B are defined using the k lowest frequency eigenvectors of the Laplacian matrix of the graph, as they preserve the global structure of graphs [92]. The node representations are obtained by multiplication of the matrix B and the adjacency matrix of the input graph. In [200], it has been shown that most common GNNs perform l_2-based graph smoothing on the graph signal in the message passing, which leads to global smoothness. Motivated by the trend filtering idea [285], Elastic GNN [179] accounts for different smoothness levels for different regions of the graph using l_1-based graph smoothing. In [364], a framelet graph convolution is proposed. This method is based on graph framelets and their transforms [365]. Framelet convolution can lower the feature and structure noises in graph representation. This method decomposes the graph into low-pass and high-pass matrices and generates framelet coefficients. Then, the coefficients are compressed by shrinkage activation, which improves the network denoising properties. **Simple Graph Convolution (SGC)** [290] is a graph convolution network that simplifies the GCN model by removing the non-linear activation functions at consecutive layers. This study theoretically proves that this model corresponds to a fixed low-pass filtering in spectral domain in which similar nodes have similar embeddings. Many other studies introduce different variants of spectral-based GNNs [144, 160, 202, 210, 277, 345, 359, 381].

(2) Spatial based. Spatial-based ConvGNNs define the graph convolution similar to applying CNNs on images. Images can be viewed as a graph such that the nodes are the pixels and the edges are the proximity of pixels. When a convolution filter applies to an image, the weighted average of the pixel values of the central node and its neighbor nodes are computed. Similarly, the spatial-based graph convolutional filters generate a node representation by aggregating the node representations of neighbors of a node. One of the advantages of spatial-based ConvGNNs is that the learned parameters of models are based on close neighbors of nodes and, therefore, can be applied on different graphs with some constraints. In contrast, spectral-based models learn filters that depend on eigenvalues of a graph Laplacian and are not directly applicable on graphs with different structures. GraphSAGE [99] is one of the early spatial-based ConvGNNs. This method generates node embeddings iteratively. The node embeddings are first initialized with node attributes. Then, a node embedding at iteration k is computed by concatenating the aggregation of the node's neighbor and the node embedding at iteration $k-1$. For example, for a

node v,

$$h^k_{N(v)} = Aggregate_k\left(h^{k-1}_u, \forall u \in N(v)\right)$$
$$h^k_v = \sigma\left(W^k.Concat\left(h^{k-1}_v, h^k_{N(v)}\right)\right),$$

where h^k and W^k are the node embedding and weight matrix at iteration k. $N(v)$ is the set of neighbors of v. GraphSAGE leverages mean, LSTM, and pooling aggregators as follows:

— Mean aggregator. The mean aggregation is similar to that of a GCN [142], which takes the mean over neighbors of a node. The difference is that a GCN includes the node representation h^{k-1}_v in the mean but GraphSAGE concatenates the node representation with the mean aggregation of neighbor nodes. This way, GraphSAGE avoids node information loss.
— LSTM aggregator. An LSTM aggregator aggregates neighbor node representations using an LSTM structure. It is important to note that LSTM preserves the order between nodes; however, there is no order among neighbor nodes. Therefore, GraphSAGE inputs a random permutation of nodes to alleviate this problem.
— Pooling aggregator. In this aggregation, each neighbor node is fed through a fully connected neural net. Then, an elementwise max operation is applied on the transformed nodes as follows:

$$Aggregate^{pool} = \max\left(\{\sigma(W_{pool}h_u + b), \forall u \in N(v)\}\right). \tag{24}$$

The above equation uses the max operator for pooling. However, the mean operator can be used as well. The pooling aggregator is symmetric and learnable. The pooling aggregation intuition is that it captures different aspects of the neighborhood set of a node.

The aggregation continues until K iterations. The model is trained using a loss function that generates similar node embeddings for nearby nodes in an unsupervised setting. The unsupervised loss can be replaced with task-specific objective functions. The **Graph Attention Network (GAT)** [266] utilizes the self-attention mechanism [265] to generate node representations. Unlike a GCN, which assigns a fixed weight to neighbor nodes, a GAT learns a weight for a neighbor depending on the importance of the neighbor node. The state of node v at layer k is formulated as follows:

$$h^k_v = \sigma\left(\sum_{u\in N(v)} \alpha^k_{vu} W^k h^{k-1}_u\right) \tag{25}$$

$$\alpha_{vu} = \frac{exp(\text{LeakyReLU}(a^T[Wh_u||Wh_v]))}{\sum_{k\in N_v} exp(\text{LeakyReLU}(a^T[Wh_u||Wh_k]))} \tag{26}$$

$$h^0_v = x_v, \tag{27}$$

where α_{vu} is the attention coefficient of node v to its neighbor u defined using a softmax function. N_v is the neighbor set of node v. W is the weight matrix and a is a weight vector. $||$ is the concatenation symbol. In addition to self-attention, a GAT's results benefit from using multi-head attention. Similar to GraphSAGE, a GAT is trained in an end-to-end fashion and outputs node representations. A **Multi-hop Attention Graph Neural Network (MAGNA)** [270] generalizes the attention mechanism in a GAT [266] by increasing the receptive fields of nodes in every layer. Stacking multiple layers of a GAT has the same effect; however, that causes the oversmoothing problem. MAGNA first computes the 1-hop attention matrix for every node and then uses the sum of powers of the attention matrix to account for multi-hop neighbors in every layer. To lower the computation cost, an approximated value for the multi-hop neighbor attention is computed. The MAGNA model aggregates the node features with attention values and passes the values through

a feed forward neural network to generate the node embeddings. The **Message Passing Neural Network (MPNN)** [86] proposes a general framework for ConvGNNs. In the MPNN framework, each node sends messages based on its states and updates its states based on messages received from its immediate neighbors. The forward pass of the MPNN has two parts: a message passing and a readout phase. In the message passing phase, a message function is utilized for information propagation and the node state is updated as follows:

$$h_v^t = U_t\left(h_v^{t-1}, \sum_{u \in N(v)} M_t\left(h_v^{t-1}, h_u^{t-1}, e_{vu}\right)\right) \tag{28}$$

$$h_v^0 = x_v, \tag{29}$$

where M_t is the message function and U_t updates the node representation. U_t, M_t are learnable functions. e_{vu} is the information of an edge (v, u). In the readout phase, the readout layer generates the graph embeddings using the updated node representations, $h_G = R(h_v^t | v \in G)$. Different ConvGNN methods can be formulated using this framework using different functions for U_t, M_t, R. The GN block [15] proposes another general framework for GNNs in which some of the GNN methods could fit in its description. A GN block learns nodes, edges, and the graph representations denoted as h_i^l, e_{ij}^l, u^l, respectively. Each GN block contains three update functions, ϕ and three aggregation functions, ρ:

$$e_{ij}^{l+1} = \phi^e\left(e_{ij}^l, h_i^l, h_j^l, u^l\right) \qquad m_i^{l+1} = \rho^{e\to v}\left(\left\{e_{ij}^{l+1}, \forall j \in N(i)\right\}\right) \tag{30}$$

$$h_i^{l+1} = \phi^v\left(m_i^{l+1}, h_i^l, u^l\right) \qquad m_V^{l+1} = \rho^{v\to u}\left(\left\{h_i^{l+1}, \forall i \in V\right\}\right) \tag{31}$$

$$u^{(l+1)} = \phi^u\left(m_E^{l+1}, m_V^{l+1}, h_i^l, u^l\right) \qquad m_E^{l+1} = \rho^{e\to u}\left(\left\{e_{ij}^{l+1}, \forall (i,j) \in E\right\}\right). \tag{32}$$

The GN assumption is that computation on a graph starts from an edge to a node and then to the entire graph. This phenomenon is formulated with update and aggregation functions: (1) ϕ^e updates the edge representations for each edge. (2) $\rho^{e\to v}$ aggregates the updated edge representations for the edges connected to each center node. (3) ϕ^v updates the node representations. (4) $\rho^{v\to u}$ aggregates node representation updates for all nodes. (5) $\rho^{e\to u}$ aggregates edge representation updates for all edges. (6) Finally, the entire graph representation is updated by ϕ^u.

GNN-FiLM [27] generates node embedding using the **feature-wise linear modulation (FiLM)** idea that was introduced in the visual question answering area [231]. Many common GNNs, such as GCN [142] and GraphSAGE [99], propagate information along edges using information from the source node of the edges. In GNN-FiLM, the target node representation transformation is computed and applied to incoming messages to generate the feature-wise modulation of the incoming messages. **Graph Random Neural Features (GRNFs)** [336] generate graph embeddings by preserving the metric structure of the graphs in the embedding space, therefore distinguishing between any pair of non-isomorphic graphs. This method is based on a family of graph neural feature maps. The graph neural feature maps are GNNs that can separate graphs. The outputs of these GNNs, which are scalar features, are then concatenated to generate the graph embedding. E(n) **Equivariant Graph Neural Network (EGNN)** [245] is a rotation, translation, and permutation equivariant GNN. These properties are fundamental in representing structures that show rotation and translation symmetric characteristics, such as molecular structures [237]. An EGNN takes as inputs a feature vector and an n-dimensional coordinates vector for each graph node along with the edge information and outputs the node embeddings. The main difference between this method and common GNNs is that the relative squared distance between a node's coordinates and neighbors has been considered in the GNN message-passing operation. The **Bilinear Graph Neural Network (BGNN)** [374] argues that the neighbors of a node can have interactions that may

affect the node representations. Therefore, it augments the aggregation of neighbors of a node by pairwise interactions of neighbor nodes. Motivated by factorization machines [108], it models the neighbors' interaction using a bilinear aggregator denoted by BA, which computes the average of pairwise multiplication of neighbor nodes of a node. Then, the convolution operator is defined as follows:

$$H^{(k)} = (1-\alpha).AGG(H^{(k-1)}, A) + \alpha.BA(H^{(k-1)}, A), \tag{33}$$

where $H^{(k)}$ is the node representation at the k-th layer and α is a trade-off parameter between two components. In [349], it is theoretically shown that all attention-based GNNs fail in distinguishing between certain structures due to ignoring the cardinality information in aggregation. Therefore, this article introduces two **cardinality-preserved attention (CPA)** models named Additive and Scaled. The formulation of the Additive model is as follows:

$$h_i^l = f^l\left(\sum_{j\in N(i)} \alpha_{ij}^{l-1} h_i^{l-1} + w^l \odot \sum_{j\in N(i)} h_i^{l-1}\right), \tag{34}$$

where the first term is the original attention formula and the second term captures the cardinality information. The Scaled model formula is

$$h_i^l = f^l\left(\psi^l(|N(i)|) \odot \sum_{j\in N(i)} \alpha_{ij}^{l-1} h_i^{l-1}\right), \tag{35}$$

where $\psi(|N(i)|)$ is a function that maps the cardinality value to a non-zero vector. Both these models improve the distinguishing power of the original attention model. A **Multi-Channel graph neural network (MuchGNN)** [372] generates graph representations using a graph pooling operation. However, instead of shrinking the graph layer by layer using graph pooling, which may result in loss of information, it shrinks the graph hierarchically. This method generates a series of graph channels at each layer and applies graph pooling on them to generate the graph representation at each layer. The final graph representation is the concatenation of graph representations at each layer. Policy-GNN [146] captures information for each node using different iterations of aggregations to capture the graph's structural information better. To that end, it uses meta-policy [339] trained by deep reinforcement learning to choose the number of aggregations per node. TinyGNN [307] proposes a small GNN with a short inference time. In order to capture the local structure of the graph, this method generates node representations by aggregating peer-aware representations of the node's neighbors. Peer-aware representations consider the interactions between peer nodes, which are neighbor nodes with the same distance from the center node. In addition, inspired by knowledge distillation [110], it proposes a **neighbor distillation strategy (NDS)** in a teacher–student network. The teacher network is a regular GNN that has access to the entire neighborhood and the student network is a small GNN that imitates the teacher network. Other spatial-based convolution GNNs include those discussed in [107, 149, 187, 217, 218, 224, 228, 268, 301, 324, 327, 334, 358, 380].

Despite the success of GNN methods, most require some supervision for training, which is costly. Self-supervised graph learning methods address this shortcoming by generating the graph embeddings without the need of the label information. These methods are categorized into generation-based, auxiliary property-based, contrast-based, and hybrid methods [183]. The generation-based methods generate embeddings by reconstructing the perturbed version of the input graph adjacency matrix or node/edge properties. For instance, a GPT-GNN [112] reconstructs node attributes and edges of a graph iteratively by maximizing the likelihood of the graph. GraphMAE [111] is another example of this category, which focuses on reconstructing features of graphs efficiently. The auxiliary property-based methods utilize extracted auxiliary properties of the graph as labels

for training the model. M3S [257] belongs to this category and uses cluster assignments as class labels for nodes.

Contrast-based methods are the major category in self-supervised graph learning, which maximize the **mutual information (MI)** between the augmented instances of a graph and the graph or graph elements, such as nodes and subgraphs. DGI [267] is a contrast-based method that maximizes the MI between a graph representation and the graph's node representations. This method is trained by maximizing the following objective function:

$$\frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(X,A)}\left[\log D(\overrightarrow{h_i},\overrightarrow{s})\right]+\sum_{j=1}^{M}\mathbb{E}_{(\tilde{X},\tilde{A})}\left[\log\left(1-D(\overrightarrow{\tilde{h}_j},\overrightarrow{s})\right)\right]\right), \tag{36}$$

where N is the number of nodes and M is the number of negative examples. $X, A, \tilde{X}, \tilde{A}$ are graph features, adjacency matrix, and the corrupted version of these matrices, respectively. $D(.,.)$ is the probability score. $\overrightarrow{s}$ is the graph representation vector, and $\overrightarrow{h_i}$ and $\overrightarrow{\tilde{h}_j}$ are the representation vectors of node i and node j obtained by summarizing a subgraph surrounding the node from the graph and the corrupted version of the graph, respectively. SimGCL [332] is another contrast-based method that generates contrastive views by adding uniform noises to node embeddings. Other examples of contrast-based methods include those discussed in [296, 332]. Hybrid methods combine two or more self-supervised objectives for the model training. For example, GMI [230] generates the node embeddings by considering two MI maximization objectives: feature MI and topology-aware MI. Feature MI is derived by comparing a representation of a node with its neighbors' features. Topology-aware MI considers the proximity of the node with its neighbors using an MI term. A survey on self-supervised graph learning methods can be found in [183].

4.3 Spatial-Temporal Graph Neural Networks (STGNNs)

STGNNs are a category of GNN that capture both the spatial and temporal properties of a graph. They model the dynamics of graphs considering the dependency between connected nodes. There are wide applications for STGNNs, such as traffic flow forecasting [36, 97, 156, 281, 348, 352], epidemic forecasting [274], and sleep stage classification [129]. For instance, in traffic prediction, the future traffic on a road is predicted taking into consideration the traffic congestion of its connected roads in previous time periods. Most of the STGNN methods fall into CNN-based and RNN-based categories that integrate the graph convolution in CNNs and RNNs, respectively.

4.3.1 RNN based. A **Graph Convolutional Recurrent Network (GCRN)** [248] is an example of RNN-based STGNN. In this method, an LSTM network is combined with the convolution operation. The GCRN has two variants. In the first variant, a CNN layer is stacked with an LSTM layer. The CNN layer extracts the features at time t and the LSTM captures the temporal behavior of nodes over time. Therefore,

$$x_t^{CNN} = CNN_G(x_t), \tag{37}$$

where x_t and x_t^{CNN} are the features and extracted features by CNN at time t. Then, x_t^{CNN} is input to the LSTM gates. For instance, for the input gate i, we have the following formula:

$$i = \sigma\left(Wx_t^{CNN} + Uh_{t-1} + b\right), \tag{38}$$

where W and U are the weights of the LSTM layer. h_{t-1}, b, and σ are the hidden state of LSTM at time $t-1$, bias, and sigmoid function, respectively. Similar formulas are used for other LSTM equations. In the second variant, the GCRN replaces the matrix multiplication operation in the LSTM with the graph convolution operation. For example, the formula for the input gate i is as

follows:

$$i = \sigma(W *_G x_t + U *_G h_{t-1} + b), \tag{39}$$

where $*_G$ is the graph convolution operator. In [281], the spatial and temporal correlations between nodes are modeled using three components: a spatial GNN layer, a GRU layer, and a transformer layer. The input to the model is a sequence of graphs over time. The spatial GNN layer captures the spatial relations between nodes in each graph. Then, a GRU and transformer layers are applied on the sequence of graphs that are output from a previous layer and capture the temporal relations between graphs over time. Other RNN-based spatial-temporal GNNs include methods in [36, 64, 83, 145, 164, 166, 171, 172, 209, 256, 274, 283, 287, 291, 299, 310, 316, 319, 321, 347].

4.3.2 CNN based. Graph WaveNet [294] is a CNN-based spatial-temporal GNN. This method takes as input a graph and feature matrices of nodes for m previous timesteps and the goal is to predict the next n feature matrices. For example, in a traffic prediction application, a feature matrix is an $N \times d$ matrix in that each row contains traffic features of a node. Each node is a sensor or a road. N is the number of nodes and d is the feature vector dimension. Graph WaveNet consists of stacked spatial-temporal layers with two building blocks: a GCN and a temporal convolution layer (Gated TCN). The model considers spatial dependencies at different temporal levels using the spatial-temporal layers. In each layer, the input X is given to the temporal convolution layer, which outputs h as follows:

$$h = g(\theta_1 \star X + b) \odot \sigma(\theta_2 \star X + c), \tag{40}$$

where θ_1, θ_2, b, and c are the model parameters. g and σ are activation functions and $\star$ is the dilated convolution operator. The dilated causal convolution captures a node's historical information with sliding over the input. For a one-dimensional input sequence x and a filter $f \in R^K$, the dilated causal convolution operation is as follows:

$$x \star f(t) = \sum_{s=0}^{K-1} f(s)x(t - d \times s), \tag{41}$$

where t is the sliding step and d is the dilation factor. In **Spatial-Temporal Fusion Graph Neural Networks (SFTGNN)** [156], instead of modeling the spatial and temporal correlations of nodes separately, a spatial-temporal fusion graph is constructed using three $N \times N$ matrices to capture three kinds of correlation for each node: (1) a spatial graph for spatial neighbors, (2) a temporal graph for nodes with similar temporal sequences, and (3) a temporal connectivity graph for connection of a node at nearby time points. The spatial-temporal fusion graph is then input to a **spatial-temporal fusion graph neural module (STFGN module)** that generates node representations. In [6, 48, 55, 62, 74, 97, 102, 129, 168, 169, 190, 194, 221, 223, 227, 253, 329, 343, 348, 351, 352], many other CNN-based spatial-temporal GNNs are proposed.

4.4 Dynamic Graph Neural Net (DGNN)

Dynamic Graph Neural Networks (DGNNs) are GNNs that model a broad range of dynamic behaviors of a graph, including adding or deleting nodes and edges over time. EvolveGCN [226] is a dynamic GNN method. The idea behind it is to use an RNN model to update weights of the GCN at each time point and capture dynamics of the graph. At each timestep t, a GCN layer is used to represent the graph at time t. In order to integrate historical information of the nodes, the initial weights for the GCN at time t are the hidden states/output of RNN-based models that take as input the weights of previous GCN models. Each RNN-based model is assigned to a separate layer of GCN models. The EvolveGCN model is trained end to end for link prediction and edge/node classification tasks. DyGNN [199] proposes a dynamic GNN method that consists of

two components: update and propagation components. These two components work in parallel to update and propagate the information of a new interaction in the graph. Let (v_s, v_g, t) represent a new directed edge that emerges between a source node v_s and a target node v_t at time t. The update component updates the two nodes v_s, v_t representations and the propagate component propagates the interaction information to *influenced nodes*, which are defined as 1-hop neighbors of the two interacting nodes.

A **Dynamic Self-Attention Network (DySAT)** [243] consists of two components, a structural block followed by a temporal block to capture the structural and temporal properties of a graph. This method defines dynamic graphs as a series of graphs over time. In order to capture the structure of the graph at each time point, the structural block, which is a variant of GAT, is applied to the graph at each time point. Then, to capture dynamic patterns of nodes, DySAT applies a temporal self-attention layer in the temporal block. The inputs to the temporal block are node representation overtime for every node v such that the node representation x_v^t attends over the historical representation of the node ($< t$) to generate the final embedding of each node. **Embedding via Historical Neighborhoods Aggregation (EHNA)** [119] aggregates the historical neighbors of a node to capture the evolution of the node in the graph. In order to capture the historical neighbors of a node, EHNA first generates k temporal random walks for each node. The transition probability of each edge in a temporal random walk depends on the weight and time of the edge. The generated random walks for each node x are first aggregated using an LSTM model to generate the walk encoding. The sequence of k walk encodings are aggregated to generate a representation for the node x. A **Temporal Graph Network (TGN)** [241] generates node embeddings for dynamic graphs that are modeled as a stream of edges. This model consists of several modules.

- *Memory*: At each time t, the TGN saves a vector $s_i(t)$ for each node i in the memory to represent the node history in a compressed format. The vector $s_i(t)$ is initialized as a zero vector and updated as more edges emerge.
- *Message Function*: Every time an edge occurs between two nodes, a message is sent to the nodes participating in the edge.
- *Message Aggregator*: A node can be involved in multiple interactions. The TGN keeps the last message in the order of time and the mean of other messages from other interactions for each node.
- *Memory Updater*: Every time an event occurs, the memory of the participating nodes is updated using a learnable memory update function such as LSTM.
- *Embedding*: The embedding module generates the temporal embeddings for each node at any time t using a learnable function h that updates the representation of each node even if the node was not involved in any interaction until that point.

In [273], a streaming GNN is modeled as $G^1, G^2, \ldots, G^T$, where $G^t = G^{t-1} + \delta G^t$ and δG^t is the changes of a graph between times $t-1$ and t. The loss of the network at time t is formulated as $L_{new} + L_{existing}$. L_{new} is the loss of parts of the graph influenced by the new changes at time t. $L_{existing}$ preserves information from previous time points. In L_{new}, the influenced nodes by new changes are replayed in the GNN model. In $L_{existing}$, the important nodes from the history are replayed. In addition, the model parameters are approximated such that they do not deviate drastically from the model parameters at the previous time. A **Temporal Graph Attention Network (TGAT)** [300] is a dynamic version of a GAT [266]. A GAT does not consider time ordering between neighbors of a node and is used for static settings. However, a TGAT assumes ordering between neighbors of a node based on the time they arrive. The assumption is that a neighbor

that occurs more recently is likely to have more influence on a node. In order to add time to the attention mechanism, a time vector is concatenated to the node feature vector. Time features that are used in TGAT are obtained based on the concepts from Bochner's Theorem and expressed by mapping the time to $\mathbb{R}^d$ as follows:

$$\phi_d(t) = \sqrt{\frac{1}{d}}[cos(w_1 t), sin(w_1 t), \ldots, cos(w_{d/2} t), sin(w_{d/2} t)], \tag{42}$$

where parameters $w_1, \ldots, w_{d/2}$ are learnable parameter. The **Causal Anonymous Walks Neural Network (CAW-N)**[282] generates temporal link embeddings by capturing the motif evolution in dynamic networks using a variant of anonymous walks [127]. This method predicts the probability of a link in the future and assumes that if motif structures of two nodes u, v interact over time, the probability of a link occurrence between u, v is higher. Therefore, this model defines set-based anonymization on the temporal random walks to capture the interaction between the motifs of the two nodes over time. Then, in order to obtain a representation of a link (u, v), all the anonymized walks for nodes u and v are encoded and aggregated using a mean or attention mechanism and are passed through an MLP to obtain the link probability. In a **Motif-preserving Temporal Shift Network (MTSN)** [184], the dynamic network is considered as a series of graph snapshots over time and two components for generating node embeddings at each time point are introduced. The first component is a **Motif Preserving Encoder (MPE)** and the second is a **Temporal shIft based on Motif preserving Encoder (TIME)**. The MPE component preserves the high-order similarity between nodes at each snapshot. First, it generates node embeddings by running a simplified GCN on the adjacency matrix and a combined motif matrix of the graph separately and adding their outputs. The combined motif matrix of a graph is obtained by weighted averaging of motif matrices of the input graph that are computed using the **Parametrized Graphlet Decomposition (PGD)** technique [4]. The TIME component considers the effect of time and is inspired by the Temporal Shift Mechanism in computer vision [170]. It shifts the node embeddings at each snapshot to capture the temporal evolution. TGR-Clique [137] is another temporal GNN that generates node and edge embeddings by preserving maximal cliques in the network. In order to generate a node embedding, this method first generates temporal random walks on the maximal cliques containing the node to capture the node's neighbors. Then, it aggregates the temporal walks of the node using **Bidirectional Long Short-Term Memory (BiLSTM)**, multi-head attention, and mean function. In [79, 98, 176, 313], several other dynamic/temporal GNNs are presented.

4.5 Real-World Applications of GNN-Based Methods

GNN methods have been deployed in production in several companies in recommendation systems, fraud detection, travel time prediction, and other types of predictions. Table 3 presents some real-world applications of GNNs. One of the main applications of GNNs in the industry is recommendation systems. Gemini [302] is an example of a recommendation system based on a GCN [142] used by the DiDiChuxing company, an online ride-hailing platform. The recommendation tasks of the company include coupon and product recommendations. Gemini deals with the heterogeneous graph of users and items by dividing it into two homogenous graphs from the users' and items' perspective called Gemini-U and Gemini-I, respectively. In Gemini-U, nodes are the users, and there is an edge between two users if they are interested in the same item, and the item will be the edge attribute. Gemini-I is generated similarly. As such, this method keeps the original user–item topology information. Then, given Gemini-U and Gemini-I, user and item representations are jointly learned to introduce the users to items' embeddings and vice versa.

Table 3. Some of the Real-World Applications of Graph Neural Networks **Deployed in Production**

The Applied Algorithm	Application
Standard GNN with MetaGradients [61]	Estimated time of travel (ETA) prediction in Google Maps
HetMatch [186]	Keyword matching for bid keyword recommendation in sponsered search platform of Alibaba Group
Category-aware GNN [236]	Review helpfulness prediction in Taobao
GNN-based tag ranking (GraphTR) [178]	Video recommendation in WeChat Top Stories
DecGCN [182]	Online recommendation system in JD.com
DHGAT [219]	Search matching in shop search in Taobao
Dynamic Heterogeneous GNN [193]	Real-time event prediction in DiDi platform
Spatial-temporal graph neural network (ConSTGAT) [75]	Travel time estimation in Baidu Maps
Heterogeneous Graph Attention Matching Network (HGAMN)[118]	Retrieving point of interests in different languages in Baidu Maps
PinSage [323]	Recommendation system at Pinterest
Gemini [302]	Online recommendation at DidiChuXing
M2GRL [275]	Recommender system at Taobao
ColdGuess [104]	Fraud detection at Amazon

GNNs have also been applied in fraud detection tasks. For instance, ColdGuess [104] is a risky listing detector used in production at Amazon. This model defines the fraud detection task as a multi-label classification task in which a classifier predicts a label for an offer based on the product complaints type. The complaint type can be normal or eight different abnormal types. In the graph of this application, nodes are sellers and products. There is an edge between a seller and a product if the seller offers that product, and the offer features are the edge features. The ColdGuess model comprises an **relational graph convolutional network (RGCN)** model [247] for generating node embeddings and an MLP model for edge embedding generation. A final classifier concatenates the nodes and edge embedding and predicts the product complaint type.

Another example of GNN applications is travel time estimation. GN blocks [15] have been used in travel time prediction in Google Maps [61]. The graph in this application is a road network in which nodes are road segments and edges are the connection between them. Each node in the graph is represented with features such as length, priority, real time and historical travel speeds and times. Given a start time and a road supersegment that contains a series of segments in the network, the model predicts the travel time labels across the supersegment in the future. For training the model, a MetaOptimizer with MetaGradients is utilized to address the variance issues resulting from using a combination of different loss functions.

4.6 The Limitation of GNNs and the Proposed Solutions

GNNs have several known limitations, such as expressive power, oversmoothing and scalability. In this section, we summarize the major articles that address these issues.

4.6.1 Expressive Power. The expressive power of a model refers to the model's ability to distinguish between different graphs. In other words, the model can map different graphs to different embeddings and similar graphs to similar embeddings. The expressive power of common GNNs such as GCN [142] and GraphSAGE [99] is bounded by the WL test and they fail to distinguish certain non-isomorphic substructures. A more powerful GNN-based embedding method called the **Graph Isomorphism Network (GIN)** is proposed in [303]. The GIN employs the sum operator instead of the mean or max operators to aggregate the neighbors of a node using the following

formula:

$$h_v^{(k)} = \text{MLP}^{(k)}\left((1+\epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}\right), \tag{43}$$

where ϵ can be a fixed or learnable parameter. GIN uses the sum aggregator, as its expressive power is higher than the mean or max operators. For instance, assume that we have two nodes v_1 and v_2. If v_1 has two equal neighbors and v_2 has three equal neighbors, the mean aggregator generates the same embedding for two nodes. The same applies to the max aggregator. However, the sum operator distinguishes different graph structures and generates different embeddings. It is also theoretically proven that GIN's expressive power is equal to the WL test. The **Identity-aware Graph Neural Network (ID-GNN)** [326] proposes a coloring mechanism as a solution for increasing the expressive power of GNNs. This model has two variants. The first variant has two components: inductive identity coloring and heterogeneous message passing. For computing the embedding for each node v, a k-hop ego network of node v is extracted and the center node of the ego network is colored. Then, for each node in the ego network of node v, the embedding is computed using a different message-passing component for each node based on color. This article proposes a fast variant of ID-GNN that augments the node features instead of coloring nodes by injecting identity information such as cycle counts for cycles that start and end in the node v. A **Nested Graph Neural Network (NGNN)** [346] suggests to encode a rooted subgraph instead of rooted subtree in common GNNs to generate node representations. It argues that rooted subtrees have limited expressiveness to represent non-tree graphs. In [157, 344], distance-based features are added to each node to increase the expressive power of GNNs, where distance vectors for each node are computed with respect to each center node. Table 4 summarizes the major studies related to increasing the expressive power of GNNs.

4.6.2 Oversmoothing. A critical known issue with GNNs is their depth limitation [158, 220]. GNN methods aggregate information from one-hop neighbors of a node in the first layer. The second layer reaches the two-hop neighbors of a node and, stacking additional layers, goes forward in the neighbors of a node similarly. After passing through multiple layers, the generated node vectors will be *oversmoothed* because the local information for each node is lost. **Graph Random Neural Network (GRAND)** [77] proposes a new framework to address the oversmoothing problem. This method augments the feature matrix of the input graph using DropNode, which randomly drops features of some of the nodes, similar to the DropEdge mechanism [240]. This augmentation helps in making nodes less sensitive to their neighbor nodes. Then, it aggregates neighbors of a node up to K-hop away using mixed-order propagation, which lowers the risk of oversmoothing. The mixed-order propagation formula is $X = \sum_{k=0}^{K} \frac{1}{K+1}\hat{A}^k\bar{X}$, where $\bar{X}$ is the augmented feature matrix. The model is trained using consistency regularization [20] also to reduce the overfitting issue in the semi-supervised setting in the case of scarce labels. In [46], it has been theoretically proved that adding two simple techniques to a GCN at each layer can overcome oversmoothing. The first technique is to construct a connection to input features to ensure that at least a fraction of node features reach the final node representation. The second technique is to add the identity matrix to the weight matrix to enforce at least the same performance as a shallow GCN. The formulation of this deep GCN is as follows:

$$H^{(l+1)} = \sigma(((1-\alpha_l)\hat{A}H^{(l)} + \alpha_l H^{(0)})((1-\beta_l)I_n + \beta_l W^{(l)})), \tag{44}$$

where α_l and β_l are the hyperparameters. $H^{(0)}$ are the initial node representations which are the node features. I_n is the identity matrix. Table 5 shows the formulation of some of the major methods that are introduced for alleviating the oversmoothing problem in GNNs.

Table 4. A Summary of Major Solutions Proposed to Increase the Expressive Power of GNNs

Authors	Algorithm	Brief summary of the solution
Xu et al. [303]	GIN	Aggregating neighbors using the sum operator
Murphy et al [213]	RP-GNN	Adding a unique node label
Morris et al. [212]	k-GNN	Performing message passing between subgraph structures instead of the node level
Maron et al. [206]	PPGN	Considering higher-order message passing
Chen et al. [50]	Ring-GNN	Using a ring of matrices in addition and multiplication
Azizian et al. [10]	FGNN	Augmenting the model with matrix multiplication
Li et al. [157]	DEGNN	Adding an extra node feature based on distance encoding
Balcilar et al. [12]	GNNML	Designing the convolution in spectral domain and masking it with a large receptive field
Barcelo et al. [13]	-	Adding local graph parameter to GNNs
Sato et al. [244]	rGIN	Adding random features to GIN [303]
Papp et al. [225]	DropGNN	Dropping some of the nodes randomly
Wang et al. [272]	PEG	Using separate channels to update the node and positional features
Wijesinghe and Wang [288]	GraphSNN	Injecting structural characteristics in the message passing
Zhang and Li [346]	NGNN	Encoding a rooted subgraph for each node instead of a rooted subtree
You et al. [326]	ID-GNN	Inductively injecting node identities either using a coloring mechanism or an augmented node feature
Dasoulas et al. [60]	CLIP	Using colors to distinguish similar node attributes
Huang et al. [125]	PG-GNN	Capturing correlation between neighboring nodes using a permutation-aware aggregation
Wijesinghe and Wang [289]	GraphSNN	Designing a local isomorphism hierarchy for node neighborhood subgraphs

4.6.3 Scalability. Another bottleneck of GNNs is scalability. In GNNs, the representations of a node's neighbors are aggregated to generate the node embeddings. Specifically, for a GNN with L layers, the neighbor aggregation computed by the matrix multiplication AH^l has the time complexity $O(LmF)$, where m is the number of neighbors and F is the hidden dimension of the model. The number of neighbors can be large in very large graphs, which lowers the GNN's training speed and increases the memory consumption. In [44, 99, 317], this problem is alleviated by sampling a subset of node neighbors. In [43], neighbors of a node are sampled at each layer independently. The Cluster-GCN [51] reduces the memory problem of the GCN by sampling a subgraph for each batch using clustering techniques and applying a graph convolution filter on the nodes in the subgraph. Some methods, such as SGC [290], remove the non-linear activation function to reduce the training time. RevGNN [152] is based on the reversible connections [88] and lowers the memory consumption of GNNs with respect to the number of layers. In this model, the feature matrix is divided into several groups, which are then input into the model to generate a group of outputs. The advantage of this model is that only the output of the last input is saved in memory for backpropagation. In [47], a **unified GNN sparsification (UGS)** framework is proposed that jointly simplifies the graph and the model to lower the GNN's inference time. The loss function of UGS is

$$L_{UGS} = L(\{m_g \odot A, X\}, m_\theta \odot \Theta) + \gamma_1 |m_g| + \gamma_2 |m_{\theta|} \tag{45}$$

where m_g, m_θ are masks for the unimportant connections in the graph and weights in GNNs. L is the cross-entropy loss and γ_1, γ_2 are hyperparameters to control the regularization of m_g, m_θ. After the training, these two masks prune the adjacency matrix and model parameters. Several

Table 5. A Summary of Major Solutions Proposed to Alleviate Oversmoothing of GNNs

Algorithm	Formula
GCN [142]	$H^{(l+1)} = \sigma(\hat{A}H^lW^l)$
APPNP [143]	$H = \sigma(\hat{A}H^lW^l), Z^{l+1} = (1-\alpha)\hat{A}Z^l + \alpha H$
JKNet-Concat [304]	$h_v = \text{concat}(h_v^1, \ldots, h_v^L)W$
GCN-PN [361]	$H^{(l+1)} = \text{TPSD}(\sigma(\hat{A}H^lW^l))$
DropEdge [240]	$H^{(l+1)} = \sigma(\hat{A}_{drop}H^lW^l)$
SGC [290]	$H = \hat{A}^LXW$
DGN-GNN [371]	$H^{(l+1)} = S^{(l+1)}H^{(l+1)} + \lambda \sum_{i=1}^{C} \gamma_i((H_i^{(l+1)} - \mu_i)/\sigma_i) + \beta_i$
DAGNN [175]	$H = \text{concat}(mlp(X), \hat{A}^1mlp(X), \ldots, \hat{A}^kmlp(X))), Z = \text{softmax}(\sigma(Hs)H)$
GRAND [77]	$Z = mlp(1/(L+1)\sum_{i=0}^{L}\hat{A}^iX_{drop})$
GCNII [46]	$H^{(l+1)} = \sigma(((1-\alpha_l)\hat{A}H^{(l)} + \alpha_lH^{(0)})((1-\beta_l)I_n + \beta_lW^{(l)}))$
GDC [103]	$H^{(l+1)} = \sigma(\sum_{i=1}^{f_l}\hat{A}_{drop}[:,i]H^l[:,i]W^l[i,:])$
PDE-GCN [72]	$h^{(l+1)} = h^l - pG^TK_l^T\sigma(K_lG_oh^l)$
GRAND-PDE [32]	$H = \psi(X(0) + \int_0^T \frac{\partial X(t)}{\partial t}dt)$
SHADOW-SAGE [337]	$h_v^{(l+1)} = \sigma(W^l.\text{concat}(h_v^l, \text{aggregate}(h_u^l, \forall u \in G_v))$
GCN+inflation [105]	$H^{(l+1)} = \text{INFLATION}^{(l+1)}(\sigma(\hat{A}H^lW^l), e)$
AdaGNN [66]	$H^{(l+1)} = H^{(l)} - \tilde{L}H^{(l)}\Phi_l$

Z is the node prediction label, H is the node representation, K_l is the convolution of the l-th layer, s is a projection vector, G_o is the discrete gradient operator on the graph, $\hat{A}_{drop}$ is the symmetric normalized adjacency matrix with certain number of edges dropped, INFLATION(.,e) = Normal(Power(Softmax(.),e)), f_l is the number of features at layer l, S^l is the clustering assignment matrix, C is number of groups, μ_i, σ_i are the mean and standard deviation of group i. γ_i, β_i, λ are hyperparameters. TPSD is a total pairwise squared distance measure, X_{drop} is the perturbed feature matrix, p is a step size, G_v is an extracted subgraph for a node v. Φ_l is a learnable parameter, $\tilde{L}$ is symmetric normalized Laplacian matrix. psi is a learnable function, X is the feature matrix. T is timestep.

other papers that studied the scalability of GNNs are [23, 45, 63, 78, 121, 126, 128, 229, 325]. Table 6 provides the complexity of GCN and some of the proposed approaches for lowering its complexity.

4.6.4 Capturing Long-Range Dependencies in Graphs. GNNs struggle to capture long-range dependencies between nodes in the graph. The reason is that broadening the GNNs' receptive field by increasing their depth encounters the oversmoothing problem in node representations. To solve this problem, in [292], a transformer module is combined with the standard GNN to capture the long-range relationships. An **Efficient infinite-depth graph neural net (EIGNN)**[174] has implicit infinite layers that can capture very long dependencies. The output of an EIGNN before the softmax layer is a limit of an infinite sequence of convolutions:

$$f(X, F, B) = B\left(\lim_{H\to\infty} Z^{(H)}\right), \tag{46}$$

where $Z^{(H)} = \gamma g(F)Z^{(H-1)}S + X$ is the output of the H-th layer. F, B are learnable weight parameters, S is the normalized adjacency matrix of the input graph, and $\gamma \in (0, 1]$. The $g(F)$ function is defined as $\frac{1}{|F^TF|+\epsilon_F}F^TF$, which is constrained to be less than one. As a result, the infinite sequence

Table 6. Time and Memory Complexities of GCN and Some of the Proposed Scalable GNN Models

Method	Solution	Time complexity	Memory complexity
GCN [142]	-	$O(LmF + LnF^2)$	$O(LnF)$
GraphSAGE [99]	Neighbor sampling	$O(ns_n^L F + ns_n^{L-1}F^2)$	$O(s_n^L bF)$
SGC [290]	Linear model	$O(nF^2)$	$O(bF)$
ClusterGCN [51]	Graph sampling	$O(LmF + LnF^2)$	$O(LbF)$
FastGCN [43]	Layer sampling	$O(Lns_l F + LnF^2)$	$O(Ls_l bF)$
LADIES [382]	Layer sampling	$O(Lns_l F + LnF^2)$	$O(Ls_l bF)$
GraphSAINT [338]	Graph sampling	$O(LbdF + LnF^2)$	$O(LbF)$
VR-GCN [44]	Graph sampling	$O(LmF + LnF^2 + s_n^L nF^2)$	$O(LnF)$
GBP [45]	Linear model	$O(LnF^2)$	$O(bF)$
RevGNN [152]	Reversible connections	$O(LmF + LnF^2)$	$O(nF)$
VQ-GNN [63]	Vector quantization	$O(LbdF + LnF^2 + LnkF)$	$O(LbF + LkF)$
BNS [317]	Neighbor sampling	$O(\tilde{s}_n^{L-1}.(s_n bF + (\delta/(1-\delta)+1).bF^2))$	$O(\tilde{s}_n^{L-1} s_n bF)$
GLT [47]	Graph sampling and model pruning	$O(Lm_g F + Lm_\theta nF^2)$	$O(LnF)$

L is the number of layers, F is the hidden dimension of the model, n is the number of nodes, m is the number of neighbors per node, b is the batch size, d is the average degree of the graph, s_n is the number of sampled nodes per node,s_l is the number of sampled nodes per layer, δ is the ratio of blocked nodes, $\tilde{s}_n = s_n \times (1-\delta)$, m_g is the number of remaining edges, m_θ is the number of remaining connections in the model, k is number of codewords.

of convolutions is convergent. A closed-form solution is derived for the EIGNN instead of using iterative solvers. In [192], the depth-wise and breadth-wise propagations are considered to capture long-range dependencies in graphs. The breadth-wise propagation is between the representation of a node with its neighbors' representations from the previous layer, which is a form of horizontal skip-connections. This model leverages a residual message function to consider edge features and alleviate the breadth-wise gradient diminishing in the model's backpropagation.

4.6.5 Catastrophic Forgetting. Catastrophic forgetting means that the neural network model may forget previously learned knowledge when trained on a new task. In [173], a **topology-aware weight preserving module (TWP)** is proposed to alleviate this issue in GNNs. This module measures the importance of the GNN's parameters after learning each task. Then, it enforces the model to remember the old parameters when learning a new task by penalizing changes to the important parameters with respect to the old tasks. In [368], an **experience replay-based model (ER-GNN)** selects some nodes from previous tasks and replays them when learning new tasks. The replayed nodes are those whose features are the closest to the mean of features in each class and have the maximum coverage and influence in model training.

4.6.6 Homophily Assumption. GNNs are based on the homophily [207] assumption, which means that nodes that are connected are similar and have the same class labels. However, it is not true in networks that nodes with opposite characteristics connect to each other. In [378], three design principles from previous methods are combined to make the new model suitable for both homophily and heterophily settings.

(1) Separating node embeddings and neighbor embeddings because mixing the node and neighbor information results in similar embeddings among a neighborhood.
(2) Considering higher-order neighborhoods to capture more relevant information from more neighbors.
(3) Combining the intermediate representations to increase the range of neighbors and information considered in node representations.

Table 7. A Summary of Major Solutions Proposed to Make GNNs Suitable for Both Homophily and Heterophily

Author	Algorithm	Brief summary of the solution
Zhu et al. [378]	H2GCN	Combining three design principles: (1) separation of ego and neighbor sampling, (2) higher-order neighbors, (3) combining intermediate representations.
Chien et al. [52]	GPR-GNN	Propagating node hidden features using generalized pagerank methods
Suresh et al. [259]	WRGNN	Generating a computation graph based on nodes' structural equivalences
Yang et al. [312]	DMP	Setting an attribute propagation weight for each edge
Zhu et al. [376]	CPGNN	Adding a compatibility matrix
Jin et al. [134]	U-GCN	Extracting three types of node embeddings from 1-hop, 2-hop, and k-nearest neighbors
Liu et al. [177]	NLGNN	Employing an attention-guided sorting of neighbor nodes
Yang et al. [314]	GPNN	Leveraging a pointer network to rank neighbor nodes
Wang et al. [277]	HOG-GCN	Incorporating a learnable homophily degree matrix into a GCN
Fang et al. [76]	Polar-GNN	Using dissimilarities between nodes in the aggregation by introducing attitude polarization
Du et al. [68]	GBK-GNN	Utilizing two kernels to capture the homophily and heterophily and selecting the appropriate one for each node pair
Li et al. [161]	GloGNN	Finding global homophily for nodes showing heterophily by learning a coefficient matrix

Similarly, it has been shown in [259] that the disassortativity of many real-world graphs can lead to the low performance of GNN models on these graphs. Therefore, this article proposes to generate a computation graph from the original graph and then run the GNN on the computation graph. The computation graph is a multi-relational graph with different types of edges between two nodes based on different levels of neighborhood similarities, such as node degrees and neighboring node degrees. In [376], a class compatibility matrix is added into GNNs to improve the performance of GNNs in graphs in which heterophily exist. This framework first estimates a prior belief of every node's class label based on the node features. Then, using a compatibility matrix H, the prior beliefs of nodes are propagated in their neighborhood. Each element H_{ij} in the matrix H is the empirical probability that nodes with class label i connect to nodes with class label j in the dataset. The compatibility matrix that can be learned in this model enables it to go beyond the homophily assumption. Table 7 summarizes some of the major papers that studied the homophily assumption in GNNs.

4.6.7 Neglecting Substructures. It has been shown in [49] that the expressive power of message-passing GNNs in detecting subgraphs of three or more nodes is limited. Therefore, a **local relation pooling (LPR)** model is proposed based on egonets. An egonet centered at a node is a subgraph consisting of nodes within a certain distance from the node in the graph. The LPR's idea is that a pattern in a graph can be found in the egonet of some node. Therefore, it generates a node representation by aggregating transformed egonets centered at the node. A **Graph Structural Kernel Network (GSKN)** [189] accounts for graph substructures such as cliques or motifs by leveraging **anonymous walk-based graph kernels (AWGK)**. Graph kernels are similarity measures

Table 8. Formulas for the Major Solutions Proposed to Help GNNs Capture Substructures

Algorithm	Formula
Geom-GCN [228]	$h_v^{l+1} = \sigma(W_l.\text{aggregate}_{i,r}((e_{(i,r)}^{v,l+1}, (i,r)))), e_{(i,r)}^{v,l+1} = \text{aggregate}(\{h_u^l \vert u \in N_i(v), \tau(z_v, z_u) = r\})$
GSN-v [24]	$h_v^{l+1} = \sigma(h_v^l, m_v^{l+1}), m_v^{l+1} = \text{aggregate}(\{h_v^l, h_u^l, x_v^V, x_u^V, e_{u,v}\})$
MPSN [22]	$h_p^{l+1} = \text{aggregate}(h_p^l, m_B^l(p), m_C^l(p), m_\downarrow^l(p), m_\uparrow^l(p))$
GraphSTONE [188]	$h_v^{l+1} = \text{aggregate}(\{\frac{R_v^T R_u}{\sum_u R_v^T R_u} h_u^l, u \in N(v)\})$
DeepLPR [49]	$H_v^{l+1} = \frac{1}{\vert S_{n_{v,t}}^{k-BFS} \vert} \sum_{\pi \in S_{n_{v,t}}^{k-BFS}} \alpha_\pi^l \odot f^l(\pi * B_{v,t}^{[ego]}(H^l))$
GSKN [189]	$h_v = \sum_{\phi \in \Phi^l(G,v)} \sigma(Z^T Z)^{-1/2} \sigma(Z^T R(\phi))$
SUBGNN [8]	$h_{x,c}^{l+1} = \sigma(W.[g_{x,c}^l; h_{x,c}^l]), g_{x,c}^l = \text{aggregate}(\{\gamma(c, A_x).a_x, \forall A_x\})$

i is a neighborhood, r is a relationship, $\tau(z_v, z_u)$ is a function that defines a relationship from node v to node u in a latent space, x_v^V are the combined structural features of node v, $e_{u,v}$ are the edge (u, v) features, h_p is the embedding of a simplex p, $m_B(p)$, $m_C(p)$, $m_\downarrow(p)$, $m_\uparrow(p)$ are the aggregation of messages from the boundary, co-boundary, lower and upper adjacent simplices of the simplex p, R_v is a row in a node-topic matrix representing the probabilities of a node v in a graph belonging to the graph's structural topics, $S_{n_{v,t}}^{k-BFS}$ is the set of permutation of subset of nodes in egonet of node v of depth t compatible with k-truncated breadth first search, α_π is a learnable normalization factor for π, f can be an MLP layer, $B_{v,t}^{[ego]}$ is the tensor representation of the egonet of node v, ϕ is an anonymous walk, $R(\phi)$ is the concatenation of the attributes of ϕ, $Z = [R(\phi_i)]_i$, Φ_l is the set of anonymous walks of length l, $h_{x,c}$ is the representation of a subgraph c, γ is a learnable similarity measure, A_x is a subgraph at channel x, a_x is the representation of A_x.

for pairwise graph comparison. This method defines an anonymous walk kernel and a random walk kernel to capture structural information in the graph. These kernels are defined based on the definition of l-walk kernels, which compute the similarity between two graphs by comparing all length l walks between every node in two graphs. Then, the GSKN formulation is derived using the kernel mapping of these two kernels. In **message-passing simplicial networks (MPSNs)** [22], the message passing is performed on simplicial complexes [214], which are a form of subgraphs consisting of several simplices. For instance, 0-simplex is a node, 1-simplex is an edge, and 2-simplex is a triangle. The representation of each simplex is computed by aggregating four types of messages received from its adjacent simplices that are present in the graph, including boundary, co-boundary, and lower and upper adjacencies. For example, an edge's boundary simplices are its vertices and the co-boundary simplices of a node are its connected edges. Finally, the global embedding for a simplicial complex is computed using a readout function on the representation of its simplices. In [8, 24, 188, 228, 261], several other GNNs with the focus on the subgraphs are presented. Table 8 demonstrates the formulas for some of these methods.

4.6.8 Over-squashing. One of the bottlenecks of GNNs is over-squashing when capturing long-range dependencies. The over-squashing is different from the oversmoothing because the over-smoothing occurs when the graph learning task only needs short-range dependencies and stacking more layers in GNN makes the node embeddings indistinguishable. However, when long-range dependencies are required and more layers are added to GNNs, the information from distant neighbors is compressed in a fixed-length vector, resulting in over-squashing and low performance of GNNs [7]. A simple solution for this problem is presented in [7], in which a fully adjacent layer is added as the last layer to a GNN model. In this layer, every two nodes are connected, which helps consider nodes beyond the nodes' local neighbors in their representation. In [262], it is proved

that negatively curved edges are the cause of over-squashing. A negative curvature occurs when an edge becomes a bridge between two sides of the graph where removing the edge disconnects them. Therefore, a curvature-based graph rewiring model is proposed to solve the over-squashing issue. This rewiring approach works by adding extra edges to support the most negatively curved edges and removing the most positively curved edges. Furthermore, the graph edit distance between the original graph and the modified one is bounded to ensure a local graph modification.

4.6.9 Design Space for GNNs. Designing effective architectures for GNNs in different tasks and datasets requires manual labor and domain knowledge. Therefore, several methods were proposed to automatically design suitable architecture for the given datasets and downstream tasks. The **Graph neural architecture search (GraphNAS)** method [82] allows the automated architecture design using reinforcement learning. In GraphNAS, a **recurrent neural network (RNN)** generates a GNN architecture that will be trained and validated on the training and validation sets. The validation result is the reward of the RNN. The RNN samples the design of each layer from a search space of different operators such as neighbor sampling, message computation, message aggregation, and readout operators. For example, a sample GNN layer is

$$[\text{first-order, gat, sum, 4, 8, relu}], \tag{47}$$

where each element in the list corresponds to neighbor sampling, message computation, message aggregation operators, number of heads, number of hidden layers, and the activation function. In [328], a general design space, task space and evaluation method for GNNs are proposed to identify the best GNN architecture for the given task quickly. The design space consists of intra-layer design, inter-layer design, and learning configuration. The intra-layer design in each layer of GNN consists of batch normalization, dropout, activation function, and aggregation function. The inter-layer design between GNN layers has four dimensions: layer connectivity, pre-process layers, message passing layers, and post-processing layers. The training configuration concerns the batch size, learning rate, optimizer, and training epochs. Then, a task similarity metric is introduced that can identify similar tasks. Finally, it develops a controlled random search evaluation to quickly find the best GNN design for the given task among many model-task combinations. In [114, 286], other methods related to the architecture design of GNNs are introduced.

4.7 GNN Libraries

For building GNN models, there exist several deep learning libraries that speed up the implementation of GNNs in practice. These libraries include PyTorch Geometric, DGL [276], Spektral, and GraphNets, which are explained below.

- **PyTorch Geometric (PyG)**[3]: This is a well-established deep learning library on graphs based on PyTorch, which has the implementation for many GNN layers, such as GraphSAGE [99], GAT [266], and GIN [303].
- **DGL**[4]: DGL is a well-known deep learning library on graphs that can be used with flexible backends, including PyTorch or TensorFlow. Several libraries have been built on top of DGL for specific purposes, such as DGL-LifeSci [155] for bioinformatics, DGL-KE [363] for knowledge graphs, and DistDGL [362] for distributed training of billion-scale graphs.
- **Spektral**[5]: This library is based on Tensorflow2 and Keras. Spektral has an easy-to-use design similar to the Keras library. However, it can be slower in training models than PyG and DGL.

[3]pytorch-geometric.readthedocs.io/en/latest
[4]dgl.ai
[5]graphneural.network

— **GraphNets**[6]: GraphNets is built on TensorFlow1 and Sonnet. However, using DGL or Spektral with TensorFlow may be easier because they are more up-to-date than GraphNets.

5 DATASETS

Graph representation learning methods are commonly evaluated in terms of their performance on a collection of datasets. In this section, we explain some of these benchmark datasets used in the literature.

— **Open Graph Benchmark (OGB)**[7]: A collection of graph datasets prepared for benchmarking the performance of different models on graphs. It consists of a diverse set of graph datasets with different sizes and domains.
— **Wikipedia**[8]: A network of wiki pages and editors as nodes. There is an edge between two nodes if an editor edits a page. The number of nodes and edges are around 9k and 157k, respectively. Edge features are user edit text. The interactions occur over 1 month.
— **Reddit**[9]: A graph between users and subreddit pages as nodes over 1 month. An edge exists between two nodes if a user posts under a page. There are around 11k nodes and 700 edges. Textual features of user posts are used as edge features.
— **Enron**[10]: This is a network of email communication between employees of the Enron corporation. Number of nodes and edges are 36k and 183k, respectively.
— **Cora**[11]: A citation network of scientific publications with 7 classes. The number of nodes and edges are 3k and 5k, respectively.
— **CiteSeer**[12]: A network of scientific publication citations from the CiteSeer digital library between around 3k nodes and 5k edges. There are 6 classes for nodes.
— **PubMed**[13]: This is a citation of publications from the PubMed database with 3 classes. This dataset has 19k nodes and 44k edges.
— **PPI**[14]: This is a protein–protein interaction network consisting of around 56k nodes and 800k edges.
— **SMS-A**[15]: A network of communication between 44k users with 548k edges over 338 days. There is an edge between two users if they exchange messages.
— **StackOverflow**[16]: An interaction network between users of the MathOverflow website. It has 14k nodes and 195k edges over 2,350 days.

6 METHODS NOT FOCUSED ON IN THIS SURVEY

6.1 Heterogeneity-Aware Graph Embedding Methods

Heterogeneous graphs contain nodes and/or edges with multiple types. The methods we discussed in this survey do not distinguish the types of nodes or edges when generating embeddings, even when applied to heterogeneous graphs. There is a stream of work on the heterogeneity-aware

[6] deepmind.com/open-source/graph-nets
[7] ogb.stanford.edu
[8] snap.stanford.edu/jodie/wikipedia.csv
[9] snap.stanford.edu/jodie/reddit.csv
[10] snap.stanford.edu/data/email-Enron.html
[11] paperswithcode.com/dataset/cora
[12] paperswithcode.com/dataset/citeseer
[13] paperswithcode.com/dataset/pubmed
[14] paperswithcode.com/dataset/ppi
[15] networkrepository.com/SMS-A.php
[16] snap.stanford.edu/data/sx-mathoverflow.html

graph embedding methods that we do not cover in this survey. Heterogeneity-aware graph embedding methods preserve the types of nodes and/or edges in generating graph embeddings. These methods are categorized into structure-preserved, attribute-assisted, application-oriented and dynamic heterogeneous graph embedding methods based on the type of information they use for generating the graph embeddings [278]. Metapath2vec [65] is a representative work from the structure-preserved category, which preserves the graph structures. This method generates meta-path-based random walks to capture the heterogeneous structural and semantic relationships between nodes. Then, it uses them to train a SkipGram model that considers the heterogeneity of nodes and edges and generates the node embeddings. HetGNN [340] belongs to the attribute-assisted category and considers the node's attributes in the node embedding. This method captures neighbors of a node by generating random walks and grouping the neighbors based on their type. Then, for each node, it encodes the features using a BiLSTM and generates the node embedding by aggregating the node's neighbors' encoded features based on the type and then all together using an attention mechanism. Other examples of heterogeneity-aware graph embedding methods include methods in [80, 113, 133, 196, 280, 315, 360, 379]. Two surveys on heterogeneity-aware graph embedding are provided in [278, 309].

6.2 Bipartite Graph-Aware Embedding Methods

Bipartite graphs are a special case of heterogeneous graphs whose nodes have two types. In addition, the edges of these graphs can only consider the interaction between two nodes with different types [85]. Bipartite graphs represent graphs in different applications, including recommendation systems. In the recommendation system, the two types of nodes are users and items. For example, SEPT [331] is a recommendation system that utilizes a user–item bipartite graph. It enhances the user–item graph with users' social relationships to improve the recommendation. Two types of relations are exploited for users, which are used as augmented views. The first view consists of triadic relationships between users. In the second view, there are edges between users that buy the same item. This method learns the user and item embeddings for the recommendation using three encoders that are jointly trained on the user–item graph and two views in a self-supervised tri-training setting. Other examples of using bipartite graphs in recommendation systems include methods in [33, 41, 115, 116, 132, 162, 295, 350]. In this survey, we do not review these types of methods. Further information about bipartite graph-aware embedding methods can be found in a related survey in [85].

6.3 Hypergraph Embedding Methods

A hypergraph is a graph in which each edge can represent the relationship between more than two nodes [367]. For instance, in an N-hop neighbor hypergraph, a hyperedge connects a user with its N-hop neighbors. An example of hypergraph embedding methods is a **multi-level graph convolutional network**(MGCN), [42], which generates hypergraph embeddings for matching the same user's accounts across different social networks. This method generates user embeddings in each social network using a convolution on the graph and refines them with a convolution on a hypergraph containing additional user information. The hypergraph convolution is obtained by replacing the information of the hypergraph in the standard GCN. The user embeddings are used to identify similar users by predicting whether there is a link between any two users across social networks. A graph with only pairwise relationships is a special hypergraph in which each edge connects two nodes. In this survey, we review methods developed for graphs with pairwise relationships. Further works on hypergraph representation learning include methods in [11, 96, 131, 258, 330].

7 OPEN ISSUES AND ONGOING RESEARCH

Graph representation learning has been a very active field of study. Although numerous techniques were developed, there are still significant issues and challenges that need to be addressed. We will discuss some of them here.

Proposing solutions for limitations of GNNs. As discussed in Section 4.6, GNNs have several limitations, such as expressive power, scalability and over-squashing. Different solutions have been proposed for these issues. However, some of them, such as over-squashing, were newly discovered and solutions to them are still preliminary, which have potential for further research. In addition, the proposed solutions for the limitations of GNNs were all for static GNNs; not much work has been done on solving the problems of dynamic and spatial-temporal GNNs. Furthermore, there might be new issues in dynamic settings that are not discovered yet and are worth studying. Finally, by applying GNNs to different real-world problems and datasets, new drawbacks of GNNs may be discovered, which will be interesting to study and solve.

Studying the theory side of GNNs. GNNs have been very successful in different applications, and researchers have tried to understand the theoretical aspects of GNNs' success. However, the theoretical analyses of GNN models in terms of optimization properties and generalization across graph sizes are less understood. In [305], the first steps in understanding the global convergence of gradient descent in GNNs are studied. In addition, in [320], it is shown that GNNs that are trained on some graph distributions cannot generalize to larger unseen graphs. Therefore, future work is needed to understand these topics fully.

Defining new GNNs by employing differential equations. Partial differential equations (PDEs) are used to model physical phenomena [25]. Recently, PDEs have been used to model the information propagation in graphs and design new GNN models [32, 72]. One of the advantages of these new GNNs is that they address the over-smoothing limitations of GNNs. Therefore, it will be interesting future work to investigate what other GNN models can be proposed using differential equations and whether these models are more powerful than other GNNs.

Including domain-knowledge in GNNs. The domain knowledge is the information about a specific problem that may not be available for a machine learning model. This knowledge can be included in the models by changing the input, loss function, and model architecture [57]. For instance, we can enrich the input data by considering additional relationships or constraints on existing relationships among the entities in the data. A few works incorporate domain knowledge into GNNs, such as [58, 59], but this area is still new and has the potential for further research.

Limited training data labels. Training neural networks with limited training data has always been challenging, which is also true for training GNNs. Solutions such as self-supervised learning, data augmentation, and contrastive learning have been proposed to solve the label scarcity problem and are employed in GNNs. However, there are still opportunities for further studying these types of learning in GNNs in both static and dynamic settings.

Transferring advances in deep learning models to GNNs. GNNs use deep learning techniques. Any advances and new models that are proposed in deep learning can be adapted in GNNs. Specifically, computer vision and natural language processing are very active areas and the new models that are developed for images, videos, and texts can be studied in graphs as well. For instance, ideas from video representation learning might be useful in dynamic graph learning because the concept of objects moving in a video's frames is similar to nodes changing over time.

Interpretability. One of the differences between representing nodes with their features, such as node degrees and centrality properties, and representing them using embedding vectors is that the embedding vectors are less understandable by a human. Therefore, developing metrics for explaining the values of node embedding vectors is necessary. There have been several works on this topic [56, 87, 138]; however, there are still opportunities to explore.

More applications. GNNs achieved great success in application in many domains, such as social networks, financial networks, and protein structures. Some of the newer applications of GNNs are in electrical power grid monitoring [239], drug overprescription prediction [342], and paper publication prediction [95]. Therefore, it is interesting to investigate the effectiveness of GNNs in other applications.

8 CONCLUSION

In this survey, we reviewed the node/graph embedding methods, which can be divided into traditional and GNN-based methods. In contrast to previous surveys on graph representation learning, we provided the literature review in both traditional and GNN-based graph embedding methods for both static and dynamic graphs and included the recent papers published up to the time of this submission. Traditional static methods generate node embedding vectors for static graphs and are categorized into factorization-based, random walk–based and non-GNN-based deep learning models. Traditional dynamic methods capture the evolving patterns in the history of nodes and are based on aggregations, random walks, non-GNN deep learning, and temporal point processes. In addition to traditional methods, GNN-based methods have achieved huge success in generating node representations. GNNs are deep learning models that generate a node representation by aggregating the node's neighbors embeddings and often use information from the particular graph mining task of interest in learning the node representations. We reviewed the general framework of GNNs and their categories, including static GNNs, spatial-temporal GNNs, and dynamic GNNs, and their real-world applications. Furthermore, we summarized nine limitations of GNNs and the proposed solutions to these limitations. These limitations are expressive power, over-smoothing, scalability, over-squashing, capturing long-range dependencies, design space, neglecting substructures, homophily assumptions, and catastrophic forgetting. Previous surveys do not provide such a summary. We also described some well-known GNN libraries, benchmark datasets, and some heterogeneity-aware, bipartite graph-aware, and hypergraph embedding methods. Finally, we discussed some of the open issues in the node representation learning field that are interesting future research directions. These future directions include proposing solutions for limitations of GNNs in both static and dynamic settings, learning GNNs with limited training data labels in both static and dynamic settings, transferring advances in deep learning to GNNs, and further studying GNN's theory side. As this field is growing fast, we hope that this up-to-date survey can provide valuable additional information for researchers working in this area.

ACKNOWLEDGMENTS

The work was supported by a Discovery Grant and a Discovery Accelerator Supplement from the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Khushnood Abbas, Alireza Abbasi, Shi Dong, Ling Niu, Laihang Yu, Bolun Chen, Shi-Min Cai, and Qambar Hasan. 2021. Application of network link prediction in drug discovery. *BMC Bioinformatics* 22 (2021), 1–21.

[2] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A. Alemi. 2018. Watch your step: Learning node embeddings via graph attention. *Advances in Neural Information Processing Systems* 31 (2018).

[3] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd International Conference on World Wide Web*. 37–48.

[4] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining*. IEEE, 1–10.

[5] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. 2019. DDGK: Learning graph representations for deep divergence graph kernels. In *The World Wide Web Conference*. 37–48.

[6] Muhammad Afif Ali, Suriya Venkatesan, Victor Liang, and Hannes Kruppa. 2021. TEST-GCN: Topologically enhanced spatial-temporal graph convolutional networks for traffic forecasting. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 982–987.

[7] Uri Alon and Eran Yahav. 2020. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*.

[8] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. 2020. Subgraph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 8017–8029.

[9] Amina Amara, Mohamed Ali Hadj Taieb, and Mohamed Ben Aouicha. 2021. Network representation learning systematic review: Ancestors and current development state. *Machine Learning with Applications* 6 (2021), 100130.

[10] Waiss Azizian et al. 2020. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*.

[11] Song Bai, Feihu Zhang, and Philip H. S. Torr. 2021. Hypergraph convolution and hypergraph attention. *Pattern Recognition* 110 (2021), 107637.

[12] Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. 2021. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*. PMLR, 599–608.

[13] Pablo Barceló, Floris Geerts, Juan Reutter, and Maksimilian Ryschkov. 2021. Graph neural networks with local graph parameters. *Advances in Neural Information Processing Systems* 34 (2021).

[14] Claudio D. T. Barros, Matheus R. F. Mendonça, Alex B. Vieira, and Artur Ziviani. 2021. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–37.

[15] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[16] Dominique Beaini, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. 2021. Directional graph networks. In *International Conference on Machine Learning*. PMLR, 748–758.

[17] Moran Beladev, Lior Rokach, Gilad Katz, Ido Guy, and Kira Radinsky. 2020. tdGraphEmbed: Temporal dynamic graph-level embedding. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 55–64.

[18] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, Vol. 14. 585–591.

[19] Kamal Berahmand, Elahe Nasiri, Yuefeng Li, et al. 2021. Spectral clustering on protein-protein interaction networks via constructing affinity matrix using attributed graph embedding. *Computers in Biology and Medicine* 138 (2021), 104933.

[20] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A. Raffel. 2019. Mixmatch: A holistic approach to semi-supervised learning. *Advances in Neural Information Processing Systems* 32 (2019).

[21] Ayan Kumar Bhowmick, Koushik Meneni, Maximilien Danisch, Jean-Loup Guillaume, and Bivas Mitra. 2020. Louvainne: Hierarchical Louvain method for high quality and scalable network embedding. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 43–51.

[22] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F. Montufar, Pietro Lio, and Michael Bronstein. 2021. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*. PMLR, 1026–1037.

[23] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2464–2473.

[24] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P. Zafeiriou, and Michael Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[25] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. 2022. Message passing neural PDE solvers. In *International Conference on Learning Representations*.

[26] Robin Brochier, Adrien Guille, and Julien Velcin. 2019. Global vectors for node representations. In *The World Wide Web Conference*. 2587–2593.

[27] Marc Brockschmidt. 2020. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning*. PMLR, 1144–1152.

[28] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations (ICLR 2014)*.

[29] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[30] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 891–900.

[31] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[32] Ben Chamberlain, James Rowbottom, Maria I. Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. 2021. GRAND: Graph neural diffusion. In *International Conference on Machine Learning*. PMLR, 1407–1418.

[33] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. 2021. Sequential recommendation with graph neural networks. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 378–387.

[34] Sudhanshu Chanpuriya and Cameron Musco. 2020. InfiniteWalk: Deep network embeddings as Laplacian embeddings with a nonlinearity. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1325–1333.

[35] Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos Tsourakakis. 2020. Node embeddings and exact low-rank representations of complex networks. *Advances in Neural Information Processing Systems* 33 (2020), 13185–13198.

[36] Cen Chen, Kenli Li, Sin G. Teo, Xiaofeng Zou, Kang Wang, Jie Wang, and Zeng Zeng. 2019. Gated residual recurrent graph neural networks for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 485–492.

[37] Chen Chen and Hanghang Tong. 2015. Fast eigen-functions tracking on dynamic graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 559–567.

[38] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C.-C. Jay Kuo. 2020. Graph representation learning: A survey. *APSIPA Transactions on Signal and Information Processing* 9 (2020).

[39] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. HARP: Hierarchical representation learning for networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[40] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. 2019. Fast and accurate network embeddings via very sparse random projection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 399–408.

[41] Hongxu Chen, Hongzhi Yin, Tong Chen, Weiqing Wang, Xue Li, and Xia Hu. 2020. Social boosted recommendation with folded bipartite network embedding. *IEEE Transactions on Knowledge and Data Engineering* 34, 2 (2020), 914–926.

[42] Hongxu Chen, Hongzhi Yin, Xiangguo Sun, Tong Chen, Bogdan Gabrys, and Katarzyna Musial. 2020. Multi-level graph convolutional networks for cross-platform anchor link prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1503–1511.

[43] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*.

[44] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*. PMLR, 942–950.

[45] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *Advances in Neural Information Processing Systems* 33 (2020), 14556–14566.

[46] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*. PMLR, 1725–1735.

[47] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. 2021. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*. PMLR, 1695–1706.

[48] Xu Chen, Yuanxing Zhang, Lun Du, Zheng Fang, Yi Ren, Kaigui Bian, and Kunqing Xie. 2020. TSSRGCN: Temporal spectral spatial retrieval graph convolutional network for traffic flow forecasting. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 954–959.

[49] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. Can graph neural networks count substructures? *Advances in Neural Information Processing Systems* 33 (2020), 10383–10395.

[50] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. 2019. On the equivalence between graph isomorphism testing and function approximation with GNNs. *Advances in Neural Information Processing Systems* 32 (2019).

[51] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.

[52] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2020. Adaptive universal generalized PageRank graph neural network. In *International Conference on Learning Representations*.

[53] Mustafa Coşkun and Mehmet Koyutürk. 2021. Node similarity-based graph convolution for link prediction in biological networks. *Bioinformatics* 37, 23 (2021), 4501–4508.

[54] Paulo Ricardo da Silva Soares and Ricardo Bastos Cavalcante Prudêncio. 2012. Time series based link prediction. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–7.
[55] Rui Dai, Shenkun Xu, Qian Gu, Chenguang Ji, and Kaikui Liu. 2020. Hybrid spatio-temporal graph convolutional network: Improving traffic prediction with navigation data. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3074–3082.
[56] Ayushi Dalmia and Manish Gupta. 2018. Towards interpretation of node embeddings. In *Companion Proceedings of the The Web Conference 2018*. 945–952.
[57] Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. 2022. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports* 12, 1 (2022), 1–15.
[58] Tirtharaj Dash, Ashwin Srinivasan, and A. Baskar. 2022. Inclusion of domain-knowledge into GNNs using mode-directed inverse entailment. *Machine Learning* 111, 2 (2022), 575–623.
[59] Tirtharaj Dash, Ashwin Srinivasan, and Lovekesh Vig. 2021. Incorporating symbolic domain knowledge into graph neural networks. *Machine Learning* 110, 7 (2021), 1609–1636.
[60] George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. 2021. Coloring graph neural networks for node disambiguation. In *Proceedings of the 29th International Conference on International Joint Conferences on Artificial Intelligence*. 2126–2132.
[61] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. 2021. ETA prediction with graph neural networks in Google Maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3767–3776.
[62] Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. 2019. Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 890–897.
[63] Mucong Ding, Kezhi Kong, Jingling Li, Chen Zhu, John Dickerson, Furong Huang, and Tom Goldstein. 2021. VQ-GNN: A universal framework to scale up graph neural networks using vector quantization. *Advances in Neural Information Processing Systems* 34 (2021).
[64] Ziluo Ding, Rui Zhao, Jiyuan Zhang, Tianxiao Gao, Ruiqin Xiong, Zhaofei Yu, and Tiejun Huang. 2022. Spatio-temporal recurrent networks for event-based optical flow estimation. *Proceedings of the AAAI Conference on Artificial Intelligence* (2022).
[65] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 135–144.
[66] Yushun Dong, Kaize Ding, Brian Jalaian, Shuiwang Ji, and Jundong Li. 2021. AdaGNN: Graph neural networks with adaptive frequency response filter. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 392–401.
[67] Lun Du, Zhicong Lu, Yun Wang, Guojie Song, Yiming Wang, and Wei Chen. 2018. Galaxy network embedding: A hierarchical community structure preserving approach. In *IJCAI*. 2079–2085.
[68] Lun Du, Xiaozhou Shi, Qiang Fu, Xiaojun Ma, Hengyu Liu, Shi Han, and Dongmei Zhang. 2022. GBK-GNN: Gated bi-kernel graph neural networks for modeling both homophily and heterophily. In *Proceedings of the ACM Web Conference 2022*. 1550–1558.
[69] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic network embedding: An extended approach for skip-gram based network embedding. In *IJCAI*, Vol. 2018. 2086–2092.
[70] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. 2011. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5, 2 (2011), 1–27.
[71] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. 2021. Implicit deep learning. *SIAM Journal on Mathematics of Data Science* 3, 3 (2021), 930–958.
[72] Moshe Eliasof, Eldad Haber, and Eran Treister. 2021. PDE-GCN: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in Neural Information Processing Systems* 34 (2021).
[73] Alessandro Epasto and Bryan Perozzi. 2019. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The World Wide Web Conference*. 394–404.
[74] Shen Fang, Qi Zhang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2019. GSTNet: Global spatial-temporal network for traffic flow prediction. In *IJCAI*. 2286–2293.
[75] Xiaomin Fang, Jizhou Huang, Fan Wang, Lingke Zeng, Haijin Liang, and Haifeng Wang. 2020. ConSTGAT: Contextual spatial-temporal graph attention network for travel time estimation at Baidu maps. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2697–2705.
[76] Zheng Fang, Lingjun Xu, Guojie Song, Qingqing Long, and Yingxue Zhang. 2022. Polarized graph neural networks. In *Proceedings of the ACM Web Conference 2022*. 1404–1413.

[77] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph random neural networks for semi-supervised learning on graphs. *Advances in Neural Information Processing Systems* 33 (2020), 22092–22103.

[78] Matthias Fey, Jan E. Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and expressive graph neural networks via historical embeddings. In *International Conferences on Machine Learning (ICML)*.

[79] Dongqi Fu and Jingrui He. 2021. SDG: A simplified and dynamic graph neural network. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2273–2277.

[80] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*. 2331–2341.

[81] Hongchang Gao and Heng Huang. 2018. Deep attributed network embedding. In *27th International Joint Conference on Artificial Intelligence (IJCAI)*.

[82] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2020. Graph neural architecture search.. In *IJCAI*, Vol. 20. 1403–1409.

[83] Yangli-ao Geng, Qingyong Li, Tianyang Lin, Lei Jiang, Liangtao Xu, Dong Zheng, Wen Yao, Weitao Lyu, and Yijun Zhang. 2019. LightNet: A dual spatiotemporal encoder network model for lightning prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2439–2447.

[84] Stavros Georgousis, Michael P. Kenning, and Xianghua Xie. 2021. Graph deep learning: State of the art and challenges. *IEEE Access* 9 (2021), 22106–22140.

[85] Edward Giamphy, Jean-Loup Guillaume, Antoine Doucet, and Kevin Sanchis. 2023. A survey on bipartite graphs embedding. *Social Network Analysis and Mining* 13, 1 (2023), 54.

[86] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272.

[87] Antonia Gogoglou, C. Bayan Bruss, and Keegan E. Hines. 2019. On the interpretability and evaluation of graph representation learning. *arXiv preprint arXiv:1910.03081* (2019).

[88] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. 2017. The reversible residual network: Backpropagation without storing activations. *Advances in Neural Information Processing Systems* 30 (2017).

[89] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* 187 (2020), 104816.

[90] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[91] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2017. DynGEM: Deep embedding method for dynamic graphs. *3rd International Workshop on Representation Learning for Graphs (ReLiG), IJCAI 2017* (2017).

[92] Denis S. Grebenkov and B.-T. Nguyen. 2013. Geometrical structure of Laplacian eigenfunctions. *SIAM Review* 55, 4 (2013), 601–667.

[93] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD*. ACM, 855–864.

[94] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. 2020. Implicit graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 11984–11995.

[95] Renchu Guan, Yonghao Liu, Xiaoyue Feng, and Ximing Li. 2021. VPALG: Paper-publication prediction with graph neural networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 617–626.

[96] Lei Guo, Hongzhi Yin, Tong Chen, Xiangliang Zhang, and Kai Zheng. 2021. Hierarchical hyperedge embedding-based representation learning for group recommendation. *ACM Transactions on Information Systems (TOIS)* 40, 1 (2021), 1–27.

[97] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 922–929.

[98] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational graph recurrent neural networks. *Advances in Neural Information Processing Systems* 32 (2019).

[99] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* 30 (2017).

[100] William L. Hamilton. 2020. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14, 3 (2020), 1–159.

[101] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[102] Liangzhe Han, Bowen Du, Leilei Sun, Yanjie Fu, Yisheng Lv, and Hui Xiong. 2021. Dynamic and multi-faceted spatio-temporal deep learning for traffic speed forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 547–555.

[103] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. 2020. Bayesian graph neural networks with adaptive connection sampling. In *International Conference on Machine Learning*. PMLR, 4094–4104.

[104] Bo He, Xiang Song, Vincent Gao, and Christos Faloutsos. 2022. ColdGuess: A general and effective relational graph convolutional network to tackle cold start cases. *arXiv preprint arXiv:2205.12318* (2022).

[105] Dongxiao He, Rui Guo, Xiaobao Wang, Di Jin, Yuxiao Huang, and Wenjun Wang. 2022. Inflation improves graph neural networks. In *Proceedings of the ACM Web Conference 2022*. 1466–1474.

[106] Tao He, Lianli Gao, Jingkuan Song, Xin Wang, Kejie Huang, and Yuanfang Li. 2020. SNEQ: Semi-supervised attributed network embedding with attention-based quantisation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4091–4098.

[107] Tiantian He, Yew Soon Ong, and Lu Bai. 2021. Learning conjoint attentions for graph neural nets. *Advances in Neural Information Processing Systems* 34 (2021), 2641–2653.

[108] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 355–364.

[109] Farzaneh Heidari and Manos Papagelis. 2020. Evolving network representation learning based on random walks. *Applied Network Science* 5, 1 (2020), 1–38.

[110] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *stat* 1050 (2015), 9.

[111] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. 2022. GraphMAE: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 594–604.

[112] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1857–1867.

[113] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of the Web Conference 2020*. 2704–2710.

[114] Zhao Huan, Yao Quanming, and Tu Weiwei. 2021. Search to aggregate neighborhood for graph neural network. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 552–563.

[115] Chao Huang, Jiahui Chen, Lianghao Xia, Yong Xu, Peng Dai, Yanqing Chen, Liefeng Bo, Jiashu Zhao, and Jimmy Xiangji Huang. 2021. Graph-enhanced multi-task learning of multi-level transition dynamics for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4123–4130.

[116] Chao Huang, Huance Xu, Yong Xu, Peng Dai, Lianghao Xia, Mengyin Lu, Liefeng Bo, Hao Xing, Xiaoping Lai, and Yanfang Ye. 2021. Knowledge-aware coupled graph neural network for social recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4115–4122.

[117] Hong Huang, Zixuan Fang, Xiao Wang, Youshan Miao, and Hai Jin. [n.d.]. Motif-preserving temporal network embedding. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI-20)*. 1237–1243.

[118] Jizhou Huang, Haifeng Wang, Yibo Sun, Miao Fan, Zhengjie Huang, Chunyuan Yuan, and Yawen Li. 2021. HGAMN: Heterogeneous graph attention matching network for multilingual POI retrieval at Baidu maps. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3032–3040.

[119] Shixun Huang, Zhifeng Bao, Guoliang Li, Yanghao Zhou, and J. Shane Culpepper. 2020. Temporal network representation learning via historical neighborhoods aggregation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1117–1128.

[120] Tong Huang, Lihua Zhou, Lizhen Wang, Guowang Du, and Kevin Lü. 2020. Attributed network embedding with community preservation. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 334–343.

[121] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in Neural Information Processing Systems* 31 (2018).

[122] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 633–641.

[123] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*. 731–739.

[124] Zexi Huang, Arlei Silva, and Ambuj Singh. 2021. A broader picture of random-walk based graph embedding. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 685–695.

[125] Zhongyu Huang, Yingheng Wang, Chaozhuo Li, and Huiguang He. 2022. Going deeper into permutation-sensitive graph neural networks. *ICML* (2022).

[126] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 675–684.

[127] Sergey Ivanov and Evgeny Burnaev. 2018. Anonymous walk embeddings. In *International Conference on Machine Learning*. PMLR, 2186–2195.
[128] Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. 2020. Redundancy-free computation for graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 997–1005.
[129] Ziyu Jia, Youfang Lin, Jing Wang, Ronghao Zhou, Xiaojun Ning, Yuanlai He, and Yaoshuai Zhao. 2020. GraphSleepNet: Adaptive spatial-temporal graph convolutional networks for sleep stage classification. In *IJCAI*. 1324–1330.
[130] Fei Jiang, Lifang He, Yi Zheng, Enqiang Zhu, Jin Xu, and Philip S. Yu. 2018. On spectral graph embedding: A non-backtracking perspective and graph approximation. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 324–332.
[131] Jianwen Jiang, Yuxuan Wei, Yifan Feng, Jingxuan Cao, and Yue Gao. 2019. Dynamic hypergraph neural networks. In *IJCAI*. 2635–2641.
[132] Bowen Jin, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. 2020. Multi-behavior recommendation with graph convolutional networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 659–668.
[133] Di Jin, Cuiying Huo, Chundong Liang, and Liang Yang. 2021. Heterogeneous graph neural network via attribute completion. In *Proceedings of the Web Conference 2021*. 391–400.
[134] Di Jin, Zhizhi Yu, Cuiying Huo, Rui Wang, Xiao Wang, Dongxiao He, and Jiawei Han. 2021. Universal graph convolutional networks. *Advances in Neural Information Processing Systems* 34 (2021).
[135] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.
[136] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research* 21, 70 (2020), 1–73.
[137] Shima Khoshraftar, Aijun An, and Nastaran Babanejad. 2022. Temporal graph representation learning via maximal cliques. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 606–615.
[138] Shima Khoshraftar, Sedigheh Mahdavi, and Aijun An. 2021. Centrality-based interpretability measures for graph embeddings. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 1–10.
[139] Shima Khoshraftar, Sedigheh Mahdavi, Aijun An, Yonggang Hu, and Junfeng Liu. 2019. Dynamic graph embedding via LSTM history tracking. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 119–127.
[140] Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. *stat* 1050 (2014), 1.
[141] Thomas N. Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[142] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR* (2017).
[143] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized PageRank. In *International Conference on Learning Representations*.
[144] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 13366–13378.
[145] Dejiang Kong and Fei Wu. 2018. HST-LSTM: A hierarchical spatial-temporal long-short term memory network for location prediction. In *IJCAI*, Vol. 18. 2341–2347.
[146] Kwei-Herng Lai, Daochen Zha, Kaixiong Zhou, and Xia Hu. 2020. Policy-GNN: Aggregation optimization for graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 461–471.
[147] Yi-An Lai, Chin-Chi Hsu, Wen Hao Chen, Mi-Yen Yeh, and Shou-De Lin. 2017. PRUNE: Preserving proximity and global ranking for network embedding. *Advances in Neural Information Processing Systems* 30 (2017).
[148] Vang Le and Vaclav Snasel. 2019. Community detection in online social network using graph embedding and hierarchical clustering. In *Proceedings of the 3rd International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'18) Volume 13*. Springer, 263–272.
[149] John Boaz Lee, Ryan A. Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. 2019. Graph convolutional networks with motif-based attention. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 499–508.
[150] A. A. Leman and B. Weisfeiler. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya* 2, 9 (1968), 12–16.
[151] Bentian Li and Dechang Pi. 2020. Network representation learning: A systematic literature review. *Neural Computing and Applications* 32, 21 (2020), 16647–16679.
[152] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021. Training graph neural networks with 1000 layers. In *International Conference on Machine Learning*. PMLR, 6437–6449.

[153] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 387–396.

[154] Meizi Li, Shuyi Lu, Lele Zhang, Yuping Zhang, and Bo Zhang. 2021. A community detection method for social network based on community embedding. *IEEE Transactions on Computational Social Systems* 8, 2 (2021), 308–318.

[155] Mufei Li, Jinjing Zhou, Jiajing Hu, Wenxuan Fan, Yangkang Zhang, Yaxin Gu, and George Karypis. 2021. DGL-LifeSci: An open-source toolkit for deep learning on graphs in life science. *ACS Omega* 6, 41 (2021), 27233–27238.

[156] Mengzhang Li and Zhanxing Zhu. 2021. Spatial-temporal fusion graph neural networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4189–4196.

[157] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems* 33 (2020), 4465–4478.

[158] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[159] Qimai Li, Xiaotong Zhang, Han Liu, Quanyu Dai, and Xiao-Ming Wu. 2021. Dimensionwise separable 2-D graph convolution for unsupervised and semi-supervised learning on graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 953–963.

[160] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive graph convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[161] Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. 2022. Finding global homophily in graph neural networks when meeting heterophily. *ICML* (2022).

[162] Yinfeng Li, Chen Gao, Xiaoyi Du, Huazhou Wei, Hengliang Luo, Depeng Jin, and Yong Li. 2022. Spatiotemporal-aware session-based recommendation with graph neural networks. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 1209–1218.

[163] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. 2018. Community detection in attributed graphs: An embedding approach. In *32nd AAAI Conference on Artificial Intelligence*.

[164] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*.

[165] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. 2016. Gated graph sequence neural networks. In *Proceedings of ICLR'16*.

[166] Xiaoyuan Liang, Guiling Wang, Martin Renqiang Min, Yi Qi, and Zhu Han. 2019. A deep spatio-temporal fuzzy neural network for passenger demand prediction. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 100–108.

[167] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology* 58, 7 (2007), 1019–1031.

[168] Nicholas Lim, Bryan Hooi, See-Kiong Ng, Xueou Wang, Yong Liang Goh, Renrong Weng, and Jagannadan Varadarajan. 2020. STP-UDGAT: Spatial-temporal-preference user dimensional graph attention network for next POI recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 845–854.

[169] Haoxing Lin, Rufan Bai, Weijia Jia, Xinyu Yang, and Yongjian You. 2020. Preserving dynamic attention for long-term spatial-temporal prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 36–46.

[170] Ji Lin, Chuang Gan, and Song Han. 2019. TSM: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7083–7093.

[171] Zhihui Lin, Maomao Li, Zhuobin Zheng, Yangyang Cheng, and Chun Yuan. 2020. Self-attention ConvLSTM for spatiotemporal prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 11531–11538.

[172] Hongbin Liu, Hao Wu, Weiwei Sun, and Ickjai Lee. 2019. Spatio-temporal GRU for trajectory classification. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1228–1233.

[173] Huihui Liu, Yiding Yang, and Xinchao Wang. 2021. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8653–8661.

[174] Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, Yiwei Wang, and Xiaokui Xiao. 2021. EIGNN: Efficient infinite-depth graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021).

[175] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 338–348.

[176] Meng Liu and Yong Liu. 2021. Inductive representation learning in temporal networks via mining neighborhood and community influences. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2202–2206.

[177] Meng Liu, Zhengyang Wang, and Shuiwang Ji. 2021. Non-local graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
[178] Qi Liu, Ruobing Xie, Lei Chen, Shukai Liu, Ke Tu, Peng Cui, Bo Zhang, and Leyu Lin. 2020. Graph neural network for tag ranking in tag-enhanced video recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2613–2620.
[179] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. 2021. Elastic graph neural networks. In *International Conference on Machine Learning*. PMLR, 6837–6849.
[180] Xin Liu, Tsuyoshi Murata, Kyoung-Sook Kim, Chatchawan Kotarasu, and Chenyi Zhuang. 2019. A general view for network embedding as matrix factorization. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. 375–383.
[181] Xueyi Liu and Jie Tang. 2021. Network representation learning: A macro and micro view. *AI Open* 2 (2021), 43–64.
[182] Yiding Liu, Yulong Gu, Zhuoye Ding, Junchao Gao, Ziyi Guo, Yongjun Bao, and Weipeng Yan. 2020. Decoupled graph convolution network for inferring substitutable and complementary items. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2621–2628.
[183] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. 2022. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2022).
[184] Zhijun Liu, Chao Huang, Yanwei Yu, and Junyu Dong. 2021. Motif-preserving dynamic attributed network embedding. In *Proceedings of the Web Conference 2021*. 1629–1638.
[185] Zhijun Liu, Chao Huang, Yanwei Yu, Peng Song, Baode Fan, and Junyu Dong. 2020. Dynamic representation learning for large-scale attributed networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1005–1014.
[186] Zongtao Liu, Bin Ma, Quan Liu, Jian Xu, and Bo Zheng. 2021. Heterogeneous graph neural networks for large-scale bid keyword matching. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3976–3985.
[187] Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. 2021. Tail-GNN: Tail-node graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1109–1119.
[188] Qingqing Long, Yilun Jin, Guojie Song, Yi Li, and Wei Lin. 2020. Graph structural-topic neural network. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1065–1073.
[189] Qingqing Long, Yilun Jin, Yi Wu, and Guojie Song. 2021. Theoretically improving graph neural networks via anonymous walk graph kernels. In *Proceedings of the Web Conference 2021*. 1204–1214.
[190] Bin Lu, Xiaoying Gan, Haiming Jin, Luoyi Fu, and Haisong Zhang. 2020. Spatiotemporal adaptive gated graph convolution network for urban traffic flow forecasting. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1025–1034.
[191] Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S. Yu, and Yanfang Ye. 2019. Temporal network embedding with micro- and macro-dynamics. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 469–478.
[192] Denis Lukovnikov and Asja Fischer. 2021. Improving breadth-wise backpropagation in graph neural networks helps learning long-range dependencies. In *International Conference on Machine Learning*. PMLR, 7180–7191.
[193] Wenjuan Luo, Han Zhang, Xiaodi Yang, Lin Bo, Xiaoqing Yang, Zang Li, Xiaohu Qie, and Jieping Ye. 2020. Dynamic heterogeneous graph neural network for real-time event prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3213–3223.
[194] Yingtao Luo, Qiang Liu, and Zhaocheng Liu. 2021. STAN: Spatio-temporal attention network for next location recommendation. In *Proceedings of the Web Conference 2021*. 2177–2185.
[195] Enxhell Luzhnica, Ben Day, and Pietro Lio. 2019. Clique pooling for graph classification. *arXiv preprint arXiv:1904.00374* (2019).
[196] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1150–1160.
[197] Tianshu Lyu, Yuan Zhang, and Yan Zhang. 2017. Enhancing the network embedding quality with structural similarity. In *Proceedings of the 2017 ACM Conference on Information and Knowledge Management*. 147–156.
[198] Jing Ma, Qiuchen Zhang, Jian Lou, Li Xiong, and Joyce C. Ho. 2021. Temporal network embedding via tensor factorization. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3313–3317.
[199] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. 2020. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 719–728.
[200] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. 2021. A unified view on graph neural networks as graph signal denoising. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1202–1211.

[201] Yao Ma and Jiliang Tang. 2021. *Deep Learning on Graphs*. Cambridge University Press.
[202] Zheng Ma, Junyu Xuan, Yu Guang Wang, Ming Li, and Pietro Liò. 2020. Path integral based convolution and pooling for graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 16421–16433.
[203] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. Oakland, CA, USA, 281–297.
[204] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. 2018. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 3762–3765.
[205] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. 2019. Dynamic joint variational graph autoencoders. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 385–401.
[206] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably powerful graph networks. *Advances in Neural Information Processing Systems* 32 (2019).
[207] Miller McPherson, Lynn Smith-Lovin, and James M. Cook. 2001. Birds of a feather: Homophily in social networks. *Annual Review of Sociology* 27, 1 (2001), 415–444.
[208] Zaiqiao Meng, Shangsong Liang, Hongyan Bao, and Xiangliang Zhang. 2019. Co-embedding attributed networks. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. 393–401.
[209] Congcong Miao, Jiajun Fu, Jilong Wang, Heng Yu, Botao Yao, Anqi Zhong, Jie Chen, and Zekun He. 2021. Predicting crowd flows via pyramid dilated deeper spatial-temporal network. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 806–814.
[210] Xupeng Miao, Nezihe Merve Gürel, Wentao Zhang, Zhichao Han, Bo Li, Wei Min, Susie Xi Rao, Hansheng Ren, Yinan Shan, Yingxia Shao, et al. 2021. DeGNN: Improving graph neural networks with graph decomposition. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1223–1233.
[211] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[212] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.
[213] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. 2019. Relational pooling for graph representations. In *International Conference on Machine Learning*. PMLR, 4663–4673.
[214] Vidit Nanda. 2021. Computational algebraic topology lecture notes. https://people.maths.ox.ac.uk/nanda/cat/TDANotes
[215] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*. 969–976.
[216] Feiping Nie, Wei Zhu, and Xuelong Li. 2017. Unsupervised large graph embedding. In *31st AAAI Conference on Artificial Intelligence*.
[217] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*. PMLR, 2014–2023.
[218] Giannis Nikolentzos and Michalis Vazirgiannis. 2020. Random walk graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 16211–16222.
[219] Xichuan Niu, Bofang Li, Chenliang Li, Rong Xiao, Haochuan Sun, Hongbo Deng, and Zhenzhong Chen. 2020. A dual heterogeneous graph attention network to improve long-tail performance for shop search in e-commerce. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3405–3415.
[220] Kenta Oono and Taiji Suzuki. 2019. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*.
[221] Boris N. Oreshkin, Arezou Amini, Lucy Coyle, and Mark J. Coates. 2021. FC-GAGA: Fully connected gated graph architecture for spatio-temporal traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[222] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1105–1114.
[223] Wentao Ouyang, Xiuwu Zhang, Li Li, Heng Zou, Xin Xing, Zhaojie Liu, and Yanlong Du. 2019. Deep spatio-temporal neural networks for click-through rate prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2078–2086.
[224] Guosheng Pan, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. 2021. Unsupervised attributed network embedding via cross fusion. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 797–805.
[225] Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. 2021. DropGNN: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021).
[226] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5363–5370.

[227] Cheonbok Park, Chunggi Lee, Hyojin Bahng, Yunwon Tae, Seungmin Jin, Kihwan Kim, Sungahn Ko, and Jaegul Choo. 2020. ST-GRAT: A novel spatio-temporal graph attention network for accurately forecasting dynamically changing road speed. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1215–1224.

[228] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2019. Geom-GCN: Geometric graph convolutional networks. In *International Conference on Learning Representations*.

[229] Jingshu Peng, Yanyan Shen, and Lei Chen. 2021. GraphANGEL: Adaptive aNd structure-aware sampling on graph NEuraL networks. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 479–488.

[230] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*. 259–270.

[231] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[232] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD*. ACM, 701–710.

[233] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. NetSMF: Large-scale network embedding as sparse matrix factorization. In *The World Wide Web Conference*. 1509–1520.

[234] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. 459–467.

[235] Zhenyu Qiu, Wenbin Hu, Jia Wu, Weiwei Liu, Bo Du, and Xiaohua Jia. 2020. Temporal network embedding with high-order nonlinear information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5436–5443.

[236] Xiaoru Qu, Zhao Li, Jialin Wang, Zhipeng Zhang, Pengcheng Zou, Junxiao Jiang, Jiaming Huang, Rong Xiao, Ji Zhang, and Jun Gao. 2020. Category-aware graph neural networks for improving e-commerce review helpfulness prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2693–2700.

[237] Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole Von Lilienfeld. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data* 1, 1 (2014), 1–7.

[238] Leonardo Ribeiro, Pedro Saverese, and Daniel Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *KDD*.

[239] Martin Ringsquandl, Houssem Sellami, Marcel Hildebrandt, Dagmar Beyer, Sylwia Henselmeyer, Sebastian Weber, and Mitchell Joblin. 2021. Power to the relational inductive bias: Graph neural networks in electrical power grids. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1538–1547.

[240] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*.

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

[242] Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi-Yadkori. 2020. A structural graph representation learning framework. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 483–491.

[243] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 519–527.

[244] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. 2021. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 333–341.

[245] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. 2021. E (n) equivariant graph neural networks. In *International Conference on Machine Learning*. PMLR, 9323–9332.

[246] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[247] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*. Springer, 593–607.

[248] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*. Springer, 362–373.

[249] Umang Sharan and Jennifer Neville. 2008. Temporal-relational classifiers for prediction in evolving domains. In *2008 8th IEEE International Conference on Data Mining*. IEEE, 540–549.

[250] Xiaobo Shen, Shirui Pan, Weiwei Liu, Yew-Soon Ong, and Quan-Sen Sun. 2018. Discrete network embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 3549–3555.
[251] Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node embedding over temporal graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 4605–4612.
[252] Joakim Skardinga, Bogdan Gabrys, and Katarzyna Musial. 2021. Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access* (2021).
[253] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. 2020. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 914–921.
[254] Han Hee Song, Tae Won Cho, Vacha Dave, Yin Zhang, and Lili Qiu. 2009. Scalable proximity estimation and link prediction in online social networks. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*. 322–335.
[255] Xiran Song, Jianxun Lian, Hong Huang, Mingqi Wu, Hai Jin, and Xing Xie. 2022. Friend recommendations with self-rescaling graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3909–3919.
[256] Jiahao Su, Wonmin Byeon, Jean Kossaifi, Furong Huang, Jan Kautz, and Anima Anandkumar. 2020. Convolutional tensor-train LSTM for spatio-temporal learning. *Advances in Neural Information Processing Systems* 33 (2020), 13714–13726.
[257] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. 2020. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5892–5899.
[258] Xiangguo Sun, Hongzhi Yin, Bo Liu, Hongxu Chen, Jiuxin Cao, Yingxia Shao, and Nguyen Quoc Viet Hung. 2021. Heterogeneous hypergraph embedding for graph classification. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 725–733.
[259] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma. 2021. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1541–1551.
[260] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th WWW*. International World Wide Web Conferences Steering Committee, 1067–1077.
[261] Erik Thiede, Wenda Zhou, and Risi Kondor. 2021. Autobahn: Automorphism-based graph neural nets. *Advances in Neural Information Processing Systems* 34 (2021).
[262] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. 2022. Understanding over-squashing and bottlenecks on graphs via curvature. *International Conference on Learning Representations* (2022).
[263] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*.
[264] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*. 539–548.
[265] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017).
[266] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.
[267] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. *ICLR (Poster)* 2, 3 (2019), 4.
[268] Clement Vignac, Andreas Loukas, and Pascal Frossard. 2020. Building powerful and equivariant graph neural networks with structural message-passing. *Advances in Neural Information Processing Systems* 33 (2020), 14143–14155.
[269] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD*. ACM, 1225–1234.
[270] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. 2021. Multi-hop attention graph neural networks. In *IJCAI*.
[271] Hao Wang, Enhong Chen, Qi Liu, Tong Xu, Dongfang Du, Wen Su, and Xiaopeng Zhang. 2018. A united approach to learning sparse attributed network embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 557–566.
[272] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. 2021. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations*.

[273] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1515–1524.

[274] Lijing Wang, Aniruddha Adiga, Jiangzhuo Chen, Adam Sadilek, Srinivasan Venkatramanan, and Madhav Marathe. 2022. CausalGNN: Causal-based graph neural networks for spatio-temporal epidemic forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence* (2022).

[275] Menghan Wang, Yujie Lin, Guli Lin, Keping Yang, and Xiao-ming Wu. 2020. M2GRL: A multi-task multi-view graph representation learning framework for web-scale recommender systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2349–2358.

[276] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[277] Tao Wang, Rui Wang, Di Jin, Dongxiao He, and Yuxiao Huang. 2022. Powerful graph convolutioal networks with adaptive propagation mechanism for homophily and heterophily. *Proceedings of the AAAI Conference on Artificial Intelligence* (2022).

[278] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S. Yu Philip. 2022. A survey on heterogeneous graph embedding: Methods, techniques, applications and sources. *IEEE Transactions on Big Data* (2022).

[279] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community preserving network embedding. In *31st AAAI Conference on Artificial Intelligence*.

[280] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*. 2022–2032.

[281] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. 2020. Traffic flow prediction via spatial temporal graph neural network. In *Proceedings of The Web Conference 2020*. 1082–1092.

[282] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2020. Inductive representation learning in temporal networks via causal anonymous walks. In *International Conference on Learning Representations*.

[283] Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S. Yu. 2017. PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs. *Advances in Neural Information Processing Systems* 30 (2017).

[284] Yaojing Wang, Guosheng Pan, Yuan Yao, Hanghang Tong, Hongxia Yang, Feng Xu, and Jian Lu. 2020. Bringing order to network embedding: A relative ranking based approach. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1585–1594.

[285] Yu-Xiang Wang, James Sharpnack, Alex Smola, and Ryan Tibshirani. 2015. Trend filtering on graphs. In *Artificial Intelligence and Statistics*. PMLR, 1042–1050.

[286] Zhili Wang, Shimin Di, and Lei Chen. 2021. AutoGEL: An automated graph neural network with explicit link information. *Advances in Neural Information Processing Systems* 34 (2021).

[287] Zhaonan Wang, Renhe Jiang, Zekun Cai, Zipei Fan, Xin Liu, Kyoung-Sook Kim, Xuan Song, and Ryosuke Shibasaki. 2021. Spatio-temporal-categorical graph neural networks for fine-grained multi-incident co-prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2060–2069.

[288] Asiri Wijesinghe and Qing Wang. 2021. A new perspective on "How graph neural networks go beyond Weisfeiler-Lehman?". In *International Conference on Learning Representations*.

[289] Asiri Wijesinghe and Qing Wang. 2022. A new perspective on "How graph neural networks go beyond Weisfeiler-Lehman?". *ICML* (2022).

[290] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International Conference on Machine Learning*. PMLR, 6861–6871.

[291] Fan Wu and Lixia Wu. 2019. DeepETA: A spatial-temporal sequential neural network model for estimating time of arrival in package delivery system. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 774–781.

[292] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. 2021. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems* 34 (2021), 13266–13279.

[293] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[294] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for deep spatial-temporal graph modeling. In *IJCAI*.

[295] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. 2021. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence* 2, 2 (2021), 109–127.

[296] Lianghao Xia, Chao Huang, Yong Xu, Jiashu Zhao, Dawei Yin, and Jimmy Huang. 2022. Hypergraph contrastive collaborative filtering. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 70–79.

[297] Wenyi Xiao, Huan Zhao, Vincent W. Zheng, and Yangqiu Song. 2020. Vertex-reinforced random walk for network embedding. In *Proceedings of the 2020 SIAM International Conference on Data Mining*. SIAM, 595–603.
[298] Yu Xie, Chunyi Li, Bin Yu, Chen Zhang, and Zhouhua Tang. 2020. A survey on dynamic network embedding. *arXiv preprint arXiv:2006.08093* (2020).
[299] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Liu, and Xiang Zhang. 2019. Spatio-temporal attentive RNN for node classification in temporal attributed graphs. In *IJCAI*. 3947–3953.
[300] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. *ICLR* (2020).
[301] Fengli Xu, Quanming Yao, Pan Hui, and Yong Li. 2021. Automorphic equivalence-aware graph neural network. *Advances in Neural Information Processing Systems* 34 (2021).
[302] Jixing Xu, Zhenlong Zhu, Jianxin Zhao, Xuanye Liu, Minghui Shan, and Jiecheng Guo. 2020. Gemini: A novel and universal heterogeneous graph information fusing framework for online recommendations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3356–3365.
[303] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks?. In *International Conference on Learning Representations*.
[304] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*. PMLR, 5453–5462.
[305] Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi. 2021. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In *International Conference on Machine Learning*. PMLR, 11592–11602.
[306] Mengjia Xu. 2021. Understanding graph embedding methods and their applications. *SIAM Rev.* 63, 4 (2021), 825–853.
[307] Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. 2020. TinyGNN: Learning efficient graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1848–1856.
[308] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. 2017. Fast network embedding enhancement via high order proximity approximation. In *IJCAI*, Vol. 17. 3894–3900.
[309] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 4854–4873.
[310] Guolei Yang, Ying Cai, and Chandan K. Reddy. 2018. Spatio-temporal check-in time prediction with recurrent neural network based survival analysis. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 2976–2983.
[311] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. Binarized attributed network embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1476–1481.
[312] Liang Yang, Mengzhe Li, Liyang Liu, Chuan Wang, Xiaochun Cao, Yuanfang Guo, et al. 2021. Diverse message passing for attribute with heterophily. *Advances in Neural Information Processing Systems* 34 (2021).
[313] Menglin Yang, Ziqiao Meng, and Irwin King. 2020. FeatureNorm: L2 feature normalization for dynamic graph embedding. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 731–740.
[314] Tianmeng Yang, Yujing Wang, Zhihan Yue, Yaming Yang, Yunhai Tong, and Jing Bai. 2022. Graph pointer neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence* (2022).
[315] Yaming Yang, Ziyu Guan, Jianxin Li, Wei Zhao, Jiangtao Cui, and Quan Wang. 2021. Interpretable and efficient heterogeneous graph convolutional network. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[316] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. 2019. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5668–5675.
[317] Kai-Lang Yao and Wu-Jun Li. 2021. Blocking-based neighbor sampling for large-scale graph neural networks. In *International Joint Conference on Artificial Intelligence*.
[318] Lin Yao, Luning Wang, Lv Pan, and Kai Yao. 2016. Link prediction based on common-neighbors for dynamic social network. *Procedia Computer Science* 83 (2016), 82–89.
[319] Junchen Ye, Leilei Sun, Bowen Du, Yanjie Fu, Xinran Tong, and Hui Xiong. 2019. Co-prediction of multiple transportation demands based on deep spatio-temporal neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 305–313.
[320] Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. 2021. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*. PMLR, 11975–11986.
[321] Peiyu Yi, Feihu Huang, and Jian Peng. 2021. A fine-grained graph-based spatiotemporal network for bike flow prediction in bike-sharing systems. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 513–521.

[322] Yuan Yin and Zhewei Wei. 2019. Scalable graph embeddings via sparse transpose proximities. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1429–1437.
[323] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.
[324] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. *Advances in Neural Information Processing Systems* 31 (2018).
[325] Minji Yoon, Théophile Gervet, Baoxu Shi, Sufeng Niu, Qi He, and Jaewon Yang. 2021. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2046–2056.
[326] Jiaxuan You, Jonathan M. Gomes-Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 10737–10745.
[327] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. In *International Conference on Machine Learning*. PMLR, 7134–7143.
[328] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 17009–17021.
[329] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 3634–3640.
[330] Chia-An Yu, Ching-Lun Tai, Tak-Shing Chan, and Yi-Hsuan Yang. 2018. Modeling multi-way relations with hypergraph embedding. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1707–1710.
[331] Junliang Yu, Hongzhi Yin, Min Gao, Xin Xia, Xiangliang Zhang, and Nguyen Quoc Viet Hung. 2021. Socially-aware self-supervised tri-training for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2084–2092.
[332] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. 2022. Are graph augmentations necessary? Simple graph contrastive learning for recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1294–1303.
[333] Wenchao Yu, Wei Cheng, Charu C. Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. NetWalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2672–2681.
[334] Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J. Kim. 2021. Neo-GNNs: Neighborhood overlap-aware graph neural networks for link prediction. *Advances in Neural Information Processing Systems* 34 (2021).
[335] Nazar Zaki, Harsh Singh, and Elfadil A. Mohamed. 2021. Identifying protein complexes in protein-protein interaction data using graph convolutional network. *IEEE Access* 9 (2021), 123717–123726.
[336] Daniele Zambon, Cesare Alippi, and Lorenzo Livi. 2020. Graph random neural features for distance-preserving graph representations. In *International Conference on Machine Learning*. PMLR, 10968–10977.
[337] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021).
[338] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. GraphSAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*.
[339] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. 2019. Experience replay optimization. In *IJCAI*.
[340] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.
[341] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. Network representation learning: A survey. *IEEE Transactions on Big Data* 6, 1 (2018), 3–28.
[342] Jianfei Zhang, Ai-Te Kuo, Jianan Zhao, Qianlong Wen, Erin Winstanley, Chuxu Zhang, and Yanfang Ye. 2021. RxNet: Rx-refill graph neural network for overprescribing detection. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2537–2546.
[343] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *31st AAAI Conference on Artificial Intelligence*.
[344] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems* 31 (2018).

[345] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI Conference on Artificial Intelligence*.

[346] Muhan Zhang and Pan Li. 2021. Nested graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021).

[347] Mingyang Zhang, Yong Li, Funing Sun, Diansheng Guo, and Pan Hui. 2021. Adaptive spatio-temporal convolutional network for traffic prediction. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1475–1480.

[348] Qi Zhang, Jianlong Chang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2020. Spatio-temporal graph structure learning for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1177–1185.

[349] Shuo Zhang and Lei Xie. 2020. Improving attention mechanism in graph neural networks via cardinality preservation. In *IJCAI: Proceedings of the Conference*, Vol. 2020. NIH Public Access, 1395.

[350] Weifeng Zhang, Jingwen Mao, Yi Cao, and Congfu Xu. 2020. Multiplex graph neural networks for multi-behavior recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2313–2316.

[351] Xiyue Zhang, Chao Huang, Yong Xu, and Lianghao Xia. 2020. Spatial-temporal convolutional graph attention networks for citywide traffic flow forecasting. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1853–1862.

[352] Xiyue Zhang, Chao Huang, Yong Xu, Lianghao Xia, Peng Dai, Liefeng Bo, Junbo Zhang, and Yu Zheng. 2021. Traffic flow forecasting with spatial-temporal graph diffusion network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 15008–15015.

[353] Xingyi Zhang, Kun Xie, Sibo Wang, and Zengfeng Huang. 2021. Learning based proximity matrix factorization for node embedding. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2243–2253.

[354] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale network embedding with iterative random projection. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 787–796.

[355] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2018. Timers: Error-bounded SVD restart on dynamic networks. In *32nd AAAI Conference on Artificial Intelligence*.

[356] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-order proximity preserved network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2778–2786.

[357] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[358] Zhen Zhang, Fan Wu, and Wee Sun Lee. 2020. Factor graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 8577–8587.

[359] Jialin Zhao, Yuxiao Dong, Ming Ding, Evgeny Kharlamov, and Jie Tang. 2021. Adaptive diffusion in graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021).

[360] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. 2021. Heterogeneous graph structure learning for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4697–4705.

[361] Lingxiao Zhao and Leman Akoglu. 2019. PairNorm: Tackling oversmoothing in GNNs. In *International Conference on Learning Representations*.

[362] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. DistDG;: Distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 36–44.

[363] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. DGL-KE: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 739–748.

[364] Xuebin Zheng, Bingxin Zhou, Junbin Gao, Yuguang Wang, Pietro Lió, Ming Li, and Guido Montufar. 2021. How framelets enhance graph neural networks. In *International Conference on Machine Learning*. PMLR, 12761–12771.

[365] Xuebin Zheng, Bingxin Zhou, Yu Guang Wang, and Xiaosheng Zhuang. 2022. Decimated framelet system on graphs and fast G-framelet transforms. *Journal of Machine Learning Research* 23, 18 (2022), 1–68.

[366] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable graph embedding for asymmetric proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[367] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in Neural Information Processing Systems* 19 (2006).

[368] Fan Zhou and Chengtai Cao. 2021. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4714–4722.

[369] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81.

[370] Jingya Zhou, Ling Liu, Wenqi Wei, and Jianxi Fan. 2022. Network representation learning: From preprocessing, feature extraction to node embedding. *ACM Computing Surveys (CSUR)* 55, 2 (2022), 1–35.

[371] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. 2020. Towards deeper graph neural networks with differentiable group normalization. *Advances in Neural Information Processing Systems* 33 (2020), 4917–4928.

[372] Kaixiong Zhou, Qingquan Song, Xiao Huang, Daochen Zha, Na Zou, and Xia Hu. 2021. Multi-channel graph neural networks. In *Proceedings of the 29th International Conference on International Joint Conferences on Artificial Intelligence*. 1352–1358.

[373] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[374] Hongmin Zhu, Fuli Feng, Xiangnan He, Xiang Wang, Yan Li, Kai Zheng, and Yongdong Zhang. 2021. Bilinear graph neural network with neighbor interactions. In *Proceedings of the 29th International Conference on International Joint Conferences on Artificial Intelligence*. 1452–1458.

[375] Hao Zhu and Piotr Koniusz. 2021. REFINE: Random range finder for network embedding. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3682–3686.

[376] Jiong Zhu, Ryan A. Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K. Ahmed, and Danai Koutra. 2021. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11168–11176.

[377] Jia Zhu, Qing Xie, and Eun Jung Chin. 2012. A hybrid time-series link prediction framework for large social network. In *International Conference on Database and Expert Systems Applications*. Springer, 345–359.

[378] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems* 33 (2020), 7793–7804.

[379] Shichao Zhu, Chuan Zhou, Shirui Pan, Xingquan Zhu, and Bin Wang. 2019. Relation structure-aware heterogeneous graph neural network. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1534–1539.

[380] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural Bellman-Ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems* 34 (2021).

[381] Chenyi Zhuang and Qiang Ma. 2018. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*. 499–508.

[382] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in Neural Information Processing Systems* 32 (2019).

[383] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2857–2866.

Received 14 June 2022; revised 20 May 2023; accepted 8 November 2023