A Comprehensive Survey of Dynamic Graph Neural Networks: Models, Frameworks, Benchmarks, Experiments and Challenges

ZhengZhao Feng, Rui Wang, TianXing Wang, Mingli Song, Sai Wu, and Shuibing He, *Member, IEEE*

***Abstract*—Dynamic Graph Neural Networks (GNNs) combine temporal information with GNNs to capture structural, temporal, and contextual relationships in dynamic graphs simultaneously, leading to enhanced performance in various applications. As the demand for dynamic GNNs continues to grow, numerous models and frameworks have emerged to cater to different application needs. There is a pressing need for a comprehensive survey that evaluates the performance, strengths, and limitations of various approaches in this domain. This paper aims to fill this gap by offering a thorough comparative analysis and experimental evaluation of dynamic GNNs. It covers 91 dynamic GNN models with a novel taxonomy, 17 dynamic GNN training frameworks, and commonly used benchmarks. We also evaluate the experimental results of ten representative dynamic GNN models and five frameworks on six datasets. Evaluation metrics focus on convergence accuracy, training efficiency, and GPU memory usage, enabling a thorough performance comparison across various models and frameworks. From the analysis and evaluation results, we identify key challenges and offer principles for future research to enhance the design of models and frameworks in the dynamic GNNs field.**

***Index Terms*—Graph neural networks, dynamic graphs, dynamic GNN models, training frameworks.**

I. Introduction

GRAPHS are essential for representing, analyzing, and predicting real-world phenomena [1], [2], [3], [4]. Graph neural networks (GNNs), such as GCN [5], GraphSAGE [6] and GAT [7], combine traditional graph computation with deep learning techniques, achieving success in tasks like link prediction [8], node classification [9], and attribute prediction [10]. Recently, dynamic graphs incorporating temporal information have unveiled hidden insights. To improve the extraction of real-world dynamic graph insights, dynamic GNNs (DGNNs), also known as temporal GNNs (TGNNs), merge temporal information with GNNs to capture structural, temporal, and contextual relationships within dynamic graphs [11], such as EvolveGCN [12], T-GCN [13], JODIE [14], and TGN [15]. These dynamic GNN models outperform static GNN models in various tasks and hold promise for applications in social network analysis [16], time series prediction [14], and traffic flow forecasting [13].

DGNNs have gained significant attention in the literature [17]. A plethora of DGNN models and frameworks have been created to address various applications [18], [19], [20], [21], [22], [23], enhance inference accuracy [12], [24], [25], [26], and improve training efficiency [27], [28], [29], [30], [31]. While several surveys on dynamic GNN models exist, they primarily focus on algorithms that fit the encoder-decoder architecture [32]. For instance, [17] review representation learning techniques for dynamic graphs, [33] explores the application of DGNN models in dynamic graph analysis, and [34] proposes a three-stage recursive temporal learning framework based on dynamic graph evolution theory. Furthermore, [35] exclusively concentrates on spatio-temporal graphs, while [36] presents a unique classification approach for DGNN models but with a limited scope. Although these surveys offer valuable insights, they are constrained to a narrow subset of the DGNN development landscape, exhibiting certain limitations:

L1: Outdated coverage of research. The surveys conducted by [17], [33], [34] are somewhat outdated, as they do not encompass the numerous new DGNN models that have emerged since their publication. On the other hand, while [35] and [36] offer more recent perspectives, the former focuses exclusively on spatio-temporal graphs, and the latter examines only a limited range of DGNN models.

L2: Absence of discussion on DGNN frameworks. Apart from a brief mention in [36] regarding TGL, none of the surveyed works discuss DGNN frameworks. Yet, these frameworks are essential for integrating models, optimizing training, and improving performance and scalability. They serve as a centralized platform for crafting various DGNN architectures, integrating

Received 16 August 2024; revised 17 March 2025; accepted 30 September 2025. Date of publication 14 October 2025; date of current version 25 November 2025. This work was supported in part by the Hangzhou Joint Fund of the Zhejiang Provincial Natural Science Foundation of China under Grant LHZSD24F020001, in part by the Zhejiang Province "Jianbing" Key R&D Project of China under Grant 2025C01010, in part by the Zhejiang Province High-Level Talents Special Support Program "Leading Talent of Technological Innovation of Ten-Thousands Talents Program" under Grant 2022R52046, and in part by the Fundamental Research Funds for the Central Universities under Grant 2021FZZX001-23 and Grant 226-2025-00067. Recommended for acceptance by A. Bonifati. *(Corresponding author: Rui Wang.)*

ZhengZhao Feng, TianXing Wang, and Shuibing He are with the Zhejiang University, Hangzhou 310027, China (e-mail: fengzhengzhao@zju.edu.cn; tianxingwang@zju.edu.cn; heshuibing@zju.edu.cn).

Rui Wang and Sai Wu are with the Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, Zhejiang University, Hangzhou 310027, China (e-mail: rwang21@zju.edu.cn; wusai@zju.edu.cn).

Mingli Song is with the Shanghai Institute for Advanced Study, Zhejiang University, Hangzhou 310027, China (e-mail: brooksong@zju.edu.cn).

Our code is made publicly available at https://github.com/fengwudi/DGNN_model_and_data. This repository contains a comprehensive collection of code links for DGNN models and frameworks discussed in our survey paper, along with commonly used graph datasets for evaluating these works. It also includes the experimental scripts and instructions utilized in our evaluation process for comparing representative dynamic GNN models and frameworks.

Digital Object Identifier 10.1109/TKDE.2025.3621291

efficient data processing, parallel computation, and tailored optimization algorithms for dynamic graph data. Moreover, DGNN frameworks facilitate advanced functionalities such as distributed training, crucial for handling large-scale datasets and real-world applications.

L3: Oversight of evaluation benchmarks. These surveyed works lack a detailed overview of evaluation benchmarks. While such benchmarks may be known to experienced researchers, newcomers may benefit from a comprehensive explanation. It is essential to introduce evaluation benchmarks thoroughly, including definitions and usage methods, to assist readers in understanding these criteria.

L4: Lack of experimental comparisons. Existing surveyed works serve as a foundational overview of DGNN models but lack experimental comparisons among these models. Such comparisons are vital for grasping performance discrepancies across various models. Challenges arise in unequivocally ranking DGNN models due to differences in datasets and evaluation metrics. Some models excel on specific datasets but may falter on others, lacking scalability. Moreover, variations in experimental settings can yield different results. Therefore, a standardized experimental setup with comprehensive and fair comparisons is necessary to tackle these issues.

L5: Meeting emerging application demands and new challenges. As technology advances, DGNN models have the potential to revolutionize various emerging fields, albeit not yet fully leveraged. Previous research has addressed existing challenges to some extent; however, the emergence of new demands brings new challenges to dynamic GNNs.

In response to the limitations mentioned above, we aim to offer a comprehensive and up-to-date overview of dynamic GNNs, encompassing recent research advancements. We introduce new classification methods to adapt to the evolving landscape of dynamic GNN models and explore existing frameworks and evaluation benchmarks. Additionally, we conduct thorough experimental comparisons of prominent DGNN models and frameworks, and examine emerging application demands and challenges in the field of dynamic GNNs. Our main contributions are outlined as follows:

C1: Comprehensive survey and novel taxonomy of DGNN models. Addressing the limitation mentioned in **L1**, we conducted an extensive survey of 82 recent DGNN models and introduced a novel classification approach (Section III). This taxonomy provides unique insights and perspectives in the current research field. By categorizing DGNN models according to their structures, features, and dynamic modeling methods, we facilitate a better understanding and comparison of the strengths and weaknesses of each model.

C2: Overview of existing DGNN frameworks. In addressing limitation **L2**, we provide a detailed overview of the current 13 DGNN frameworks, exploring their features and improvements in model optimization (Section IV). Our focus on the flexibility, scalability, and performance of these frameworks highlights their fundamental attributes.

C3: Introduction of evaluation benchmarks for DGNN. For **L3**, we present a diverse set of commonly used evaluation graph datasets and metrics for the models discussed (Section V). This extensive coverage aims to facilitate comprehensive evaluations and enhance reproducibility in various experiments.

TABLE I
LIST OF NOTATIONS

Notation	Meaning
V, $\|V\|$	Edges sets of graphs, the number of edges
E, $\|E\|$	Nodes sets of graphs, the number of nodes
X	Feature embedding of graph
$G = (V, E, X)$	Static graph structure
$A \in \mathbb{R}^{\|V\| \times \|V\|}$	Adjacency matrix of graph $G = (V, E, X)$
$Edge_{index} \in \mathbb{R}^{2 \times \|E\|}$	Edge index matrix of graph $G = (V, E, X)$
$T_n = [t_1 : t_n]$	Timestamp divided into n time intervals
$G_t = (V_t, E_t, X_t)$	Dynamic graph snapshot at time t
$G_T = (G_{t_1}, G_{t_2}, ..., G_{t_n})$	DTDG with a sequence of graph snapshots within T_n
$\varepsilon = (i, j, t)$	Edge established at time t from node i to node j
$G_T = (\varepsilon_{t_1}, \varepsilon_{t_2}, ..., \varepsilon_{t_n})$	CTDG with a stream of events
S, $\|S\|$	The set, the number of sets
TOP_K	Number of top k hits
H_t	Information of time t
λ	Intensity function
N_i	The neighbor of node i
$time\_partition_x(\cdot)$	Divide the graph into snapshots at x time intervals
$batch(\cdot)$	Divide the graph into batch
$encoder(\cdot)$	The encoder function
$rank_i$	The average recommended ranking

C4: Experimental comparison of selected works. To address **L4**, we conduct a comprehensive experimental comparison of various DGNN models and frameworks under consistent experimental settings, datasets, and metrics (Section VI). We evaluate the training accuracy, efficiency, and memory usage of these models and frameworks. We also assess multi-GPU scalability within the frameworks, examining their performance and efficiency in handling increased computational loads.

C5: Analysis of challenges in DGNN. To address limitation **L5**, we analyze the new challenges encountered by DGNN and suggest potential research directions for practitioners (Section VII).

II. Background

In this section, we first explore the different application scenarios and learning tasks related to dynamic graphs. Then, we discuss the learning techniques utilized for graph structures and time series data. Finally, we will describe the workflow of dynamic GNNs. The main notations of this paper can be found in Table I.

A. Applications of Dynamic Graph Learning

1) Dynamic Graph Scenarios: Real-world graphs display dynamic characteristics and are applied across diverse domains due to their versatility, including:

Temporal Interaction Graphs in Social Networks: Temporal interaction graphs capture the evolving relationships and interactions between social network users over time [14], offering insights into social dynamics and network evolution.

Real-Time Transaction Graphs in E-Commerce: Transaction graphs model the flow of transactions between users and products in real-time [37], facilitating fraud detection, recommendation systems, and personalized marketing strategies.

Spatio-Temporal Graphs (STG): These graphs integrate spatial and temporal dimensions, with nodes representing spatial locations and edges denoting spatial relationships [38], [39].

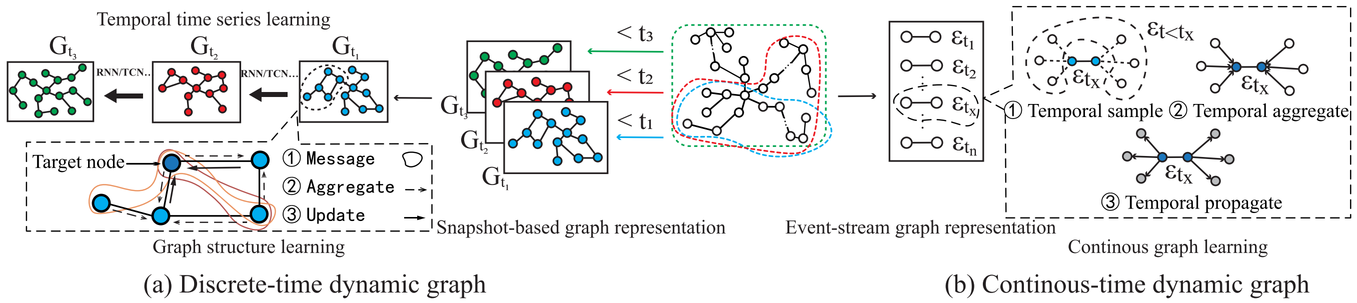
Temporal time series learning
RNN/TCN...
Target node
① Message
② Aggregate
③ Update
Graph structure learning
Snapshot-based graph representation
Event-stream graph representation
① Temporal sample
② Temporal aggregate
③ Temporal propagate
Continous graph learning
(a) Discrete-time dynamic graph
(b) Continous-time dynamic graph

Fig. 1. A toy example of dynamic graph representation and learning.

They enable the analysis of movement patterns, traffic flow, and environmental changes, aiding urban planning, transportation management, and environmental monitoring. Temporal information for nodes and edges can enhance the understanding of time-dependent spatial relationships.

Temporal Knowledge Graphs (TKG): Knowledge graphs depict structured information, with entities as nodes and relationships as edges [40]. Adding a temporal dimension in the triplets, TKGs allow to track the changes in entities and relationships over time, essential for applications like online social networks and trend analysis.

Temporal Citation Graphs (TCG): TCGs track the evolution of citations and references in scholarly publications over time, enabling analyses of research trends, influence dynamics, and knowledge dissemination patterns within academic fields [41].

2) Learning Tasks in Dynamic Graphs: Dynamic graph learning is instrumental in addressing a wide array of tasks within the aforementioned application domains, including:

Link Prediction: Involves predicting the likelihood of connections between two nodes in a network that do not yet have edges based on existing network nodes and structure [42]. In dynamic graph learning, link prediction focuses on forecasting the probability of edges appearing at a specific time.

Node Classification: Entails assigning labels to nodes with unknown labels in a graph by utilizing the connections between nodes and a limited number of labeled nodes [12]. This task aims to predict the labels of nodes at a given time.

Other Tasks: Other tasks include temporal node embedding for capturing temporal dynamics and structural changes in the graph, temporal graph embedding for learning graph representations at different time steps, and event prediction for forecasting future incidents within dynamic graphs [33].

B. Representation of Dynamic Graphs

1) Graph Structure Representation: We now delve into the evolution of graph structure representation, transitioning from traditional static graphs to dynamic graphs that capture temporal changes and evolving relationships over time.

Static Graph Representation: Static graphs are typically represented by nodes, edges, and features: $G = (V, E, X)$, where V is the node set, E is the edge set, and X represents the feature embeddings of the graph. An adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$ is commonly used to depict a static graph. In the matrix, a value of 1 at $A[i][j]$ indicates an edge from node i to node j. An edge index $Edge_{index} \in \mathbb{R}^{2\times|E|}$ is also utilized for graph data, with each column representing an edge in the graph. For example, nodes $Edge_{index}[0][i]$ and $Edge_{index}[1][i]$ define the ith edge.

Dynamic Graph Representation: Dynamic graphs add a temporal dimension to static graphs. At the time t, the graph is represented as $G_t = (V_t, E_t, X_t)$, with V_t, E_t, and X_t representing nodes, edges, and features at that time. Two common approaches to dynamic graph representation are *discrete-time dynamic graphs (DTDG)* and *continuous-time dynamic graphs (CTDG)*. The difference between DTDG and CTDG is illustrated in Fig. 1. In DTDG, the timestamp $T_n = [t_1 : t_n]$ is divided into n time intervals, and the dynamic graph is represented as a sequence of graph snapshots within T_n denoted as $G_T = (G_{t_1}, G_{t_2}, \ldots, G_{t_n})$. Each G_{t_i} captures the graph structure up to time t_i. On the other hand, in CTDG, graph information is treated as an event stream $G_T = (\varepsilon_{t_1}, \varepsilon_{t_2}, \ldots, \varepsilon_{t_n})$. An edge created at time t from node i to node j is represented as $\varepsilon_t = (i, j, t)$. CTDG maintains a single graph structure at any given time t by incorporating all event stream data to form G_t.

2) Temporal Time Series Representation: In the context of dynamic graphs, temporal time series data can be represented using explicit or implicit time methods. These two approaches provide different ways of capturing and modeling the temporal dimension of dynamic graphs over time.

Explicit Time Representation: Explicit time involves incorporating time as a separate feature in the model for computation. This approach integrates time series data directly into the model as an input feature, influencing the model's calculations and allowing it to leverage time information for tasks such as prediction or learning.

Implicit Time Representation: Implicit time signifies that the model understands the temporal progression within its structure without explicitly treating time as an input feature. Instead of requiring specific time values, the implicit models learn temporal relationships through the sequential arrangement of data. This enables the model to capture temporal evolution from the temporal data without emphasizing timestamps or explicit time features.

The difference between explicit events and implicit time is depicted in Fig. 2. Explicit time focuses on the timestamps of each edge in the graph, whereas implicit time looks at the sequential order of events. For example, implicit time only requires recognizing that snapshot 1 precedes snapshot 2. Generally, DTDG models often rely on implicit time, where time information is indirectly learned through the evolution of snapshots

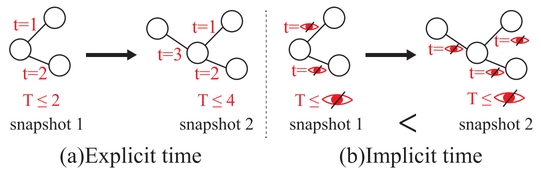

Fig. 2. The difference between explicit and implicit time.

over time. Conversely, CTDG models highlight explicit time by directly encoding timestamps in the model to capture the exact timing of interactions

C. Learning of Dynamic Graphs

1) Graph Structure Learning: Diverse methods have been developed to handle various types of graph structures, ranging from neural networks, attention mechanisms to random walks:

Graph Neural Networks (GNNs): GNNs are neural network models tailored for processing graph data, allowing for the effective capture of intricate relationships within graph structures and enabling node and graph learning and inference. A fundamental aspect of GNN design is the utilization of *message passing mechanisms*, which typically involves three key steps: *message*, *aggregate*, and *update*. Specifically, each vertex first collects feature embedding messages from its neighboring vertices then aggregates the collected feature messages using an aggregation function, and finally updates the vertex's feature embedding using neural network model.

Structural Attention (SA): Attention-based graph learning utilizes attention mechanisms in GNNs to prioritize information flow between nodes in a graph. This allows the model to focus on key nodes or edges dynamically, adjusting their importance during message passing and feature aggregation.

Random Walk: Random walk-based graph learning employs random walk algorithms on graphs to capture structural relationships between nodes. In this method, walkers move between nodes based on predefined rules (e.g., edge probabilities) to explore the graph. By performing several random walks and examining the sequences of nodes visited, we can create node embeddings that capture the local graph structure and connectivity patterns.

2) Temporal Time Series Learning: Next, we introduce the learning method for temporal time series information:

Recurrent Neural Networks (RNN): RNN is a classic neural network architecture for processing sequential data like time series and texts. It excels at capturing sequence context and adapting to different sequence lengths. Advanced RNN variations like long short-term memory (LSTM) [43] and gated recurrent unit (GRU) [44] address challenges like gradient vanishing and exploding. In addition, special RNNs such as Echo state network (ESN) [45] are also well suited for machine learning tasks involving time series data in certain scenarios.

Temporal Point Process (TPP): TPP is a statistical model used to analyze event patterns over time, focusing on event occurrence moments rather than event count or intensity [46]. The conditional intensity function $\lambda(t)$ describes the intensity of future events based on historical event information H_t before time t.

Temporal Convolutional Network (TCN): TCN [47] is a deep learning model for time series data modeling. Unlike RNNs, TCN employs a convolutional neural network structure to capture dependencies in time series, effectively capturing local dependencies within sequential data.

Temporal Attention (TA): Temporal attention is utilized for processing temporal data, enabling the DGNN models to assign varying importance to information at different time steps when handling sequential data.

Time Encoding: Time encoding involves representing temporal information as features for model training. One common approach is using parameterized Fourier features [48].

D. Dynamic Graph Neural Networks

1) DGNN Models on DTDG and CTDG: In Section II-B, we explored how dynamic graphs are represented using DTDG or CTDG, leading to distinct DGNN models. The training differences for DGNN models in DTDG and CTDG are depicted in Fig. 1. While DTDG and CTDG vary in terms of snapshots and event streams, the key contrast lies in the granularity of time representation and learning: coarse-grained time in DTDG and fine-grained time in CTDG. In DTDG, a fixed time interval like a month or a year is often selected to partition the graph into multiple snapshots when edges are added over time. Conversely, CTDG spans an infinite time range, capturing all event streams in a single snapshot graph with precise time ordering and sequential event processing.

2) Generous Definition of DGNN: For a time-varying graph structure G_t, the first step is to partition it into n_1 snapshots using (1), where $time\_partition_x(\cdot)$ denotes the division of the graph by a time interval $x \in (0, +\infty)$.

$$snapshots_{1,..,n_1} = time\_partition_x(G_t) \tag{1}$$

Given the possibility of a snapshot being too large, it may be necessary to further divide each snapshot into n_2 batches:

$$batches_{1,..,n_2} = batch(snapshot_i), where \;\; i = 1, \ldots, n_1 \tag{2}$$

Finally, a unified encoder function $encoder(\cdot)$ is used to process both structural and temporal information, generating the final output graph data G'_t:

$$G'_t = (V_t, E_t, X'_t) = encoder(batches_i), where \;\; i = 1, .., n_2 \tag{3}$$

III. Taxonomy of Dynamic GNN Models

This section offers a detailed overview of the most recent DGNN models, covering 53 DTDG models and 38 CTDG models. We introduce a new classification method that categorizes these dynamic GNN models according to their structural features, use of methods, and dynamic modeling techniques.

A. Discrete-Time Dynamic Graph Models

DTDG models are specifically designed for analyzing discrete-time dynamic graphs. These models can be classified

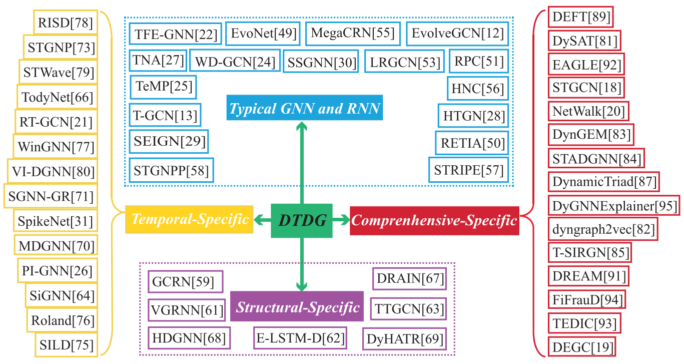

Fig. 3. The taxonomy for DTDG models in our survey.

into four groups based on their methods for handling graph structure and temporal data, as outlined in Fig. 3. **Typical GNN and RNN models** utilize GNNs for processing graph structure data and RNNs for temporal time series data. Other DTDG models fall into the categories of **structural-specific**, **temporal-specific**, and **comprehensive-specific models**, depending on their reliance on non-GNN methods for graph structure processing, non-RNN methods for temporal data processing, or neither GNN nor RNN, respectively.

1) Typical GNN and RNN Models: The conventional approach of utilizing GNN for processing graph structures and RNN for handling temporal information is considered the most intuitive and prevalent method in DGNN modeling.

Enhancing Accuracy. WD-GCN [24] was one of the pioneering algorithms that merged GNN and RNN for dynamic graph processing, combining GCN and LSTM models. After that, several typical GNN and RNN models have been developed to enhance accuracy compared to previous methods. EvolveGCN [12] addresses the challenge of frequent changes in the node set by introducing a technique to adapt graph convolutional network (GCN) models over time, without relying on node embeddings. This method captures the dynamics of graph sequences by evolving the GCN parameters using RNNs, with variations in architecture like LSTM [43] and GRU [44]. TeMP [25] incorporates temporal message passing for improved predictions, blending the structural aspects of Relational GCN and GRU. EvoNet [49] integrates graph-level propagation with GNN and RNN components to comprehensively capture temporal graph information. RETIA [50] considers adjacent relationships in addition to aggregating neighboring entities, leading to a more holistic learning approach. Incorporating GNN, GRU, and novel units RCU and PCU, RPC [51] explores relational correlations and periodic patterns, enhancing the model's expressiveness.

Improving Training Efficiency and Scalability. TNA [27] enhances training stability beyond traditional GCN and GRU structures. HTGN [28] maps the temporal graph onto hyperbolic space and introduces specific modules to constrain the model, ensuring stability and generalization of embeddings. SEIGN [29] utilizes a three-part structure involving GCN-like message passing, GRU parameter adjustments over time, and a self-attention mechanism inspired by transformers [52] for learning the final representation. SSGNN [30] utilizes echo state networks (ESN) [45] to enhance scalability for large networks. This strategy enhances scalability and efficiency, enabling effective training on large-scale graphs.

Specialized Applications. Several typical models are tailored for specific applications or domains. T-GCN [13] is a traffic prediction method based on neural network. The model uses GCN to learn complex topology, GRU is used to learn the dynamic changes of traffic data to capture time dependence, which can capture both spatial and temporal dependence, and obtain spatio-temporal correlation from traffic data. LRGCN [53] incorporates LSTM based on Relational R-GCN [54] for path fault prediction. MegaCRN [55] introduces the GCRN unit for processing spatio-temporal graphs, utilizing a hybrid architecture of GCN and GRU. TFE-GNN [22] processes graph structure using multi-layer GNNs. HNC [56] manages satellite communication status within a satellite network, aiding in the development of a global satellite joint program. STRIPE [57] is the first to incorporate memory networks for anomaly detection in spatio-temporal graphs. STGNPP [58] combines GCN and transformer for predicting traffic congestion time using neural process priors (NPP).

2) Structural-Specific Models: This category of DTDGs utilizes non-GNN methods for processing graph structural data and still uses RNN for temporal time series data processing.

Enhancing Accuracy. Several structural-specific models focus on enhancing accuracy. GCRN [59] combines GraphCNN [60] to learn spatial structure, and RNN is used to find dynamic patterns. By simultaneously utilizing graph spatial and dynamic information about the data, the accuracy and learning speed can be improved. VGRNN [61] builds upon GCRN by transitioning to GRNN and introducing the dynamic graph auto-encoder model. E-LSTM-D [62] is the first to utilize LSTM and an encoder-decoder architecture for link prediction in dynamic networks. TTGCN [63] introduces a dynamic graph representation learning method based on k-truss decomposition, effectively capturing multi-scale topological structure information in graphs. SiGNN [64] integrates the temporal dynamics of SNN [65] with the graph processing power of GNN through a novel temporal activation mechanism. TodyNet [66] introduces a hierarchical temporal pooling to surpass the limitations of flat pooling and improve performance.

Improving Training Efficiency. DRAIN [67] has constructed a recurrent graph generation scenario to represent a dynamic GNN that learns across different time points. It captures the time drift of model parameters and data distributions, and can predict future models in the absence of future data.

Specialized Applications. Some structural-specific models are specifically designed to effectively handle the heterogeneous graphs. HDGNN [68] employs multi-head attention and random walk techniques for processing structural information, whereas DyHATR [69] integrates a hierarchical attention mechanism to learn heterogeneous information. MDGNN [70] comprehensively models the complex relationships between stocks and their temporal dynamics, leveraging the Transformer architecture to effectively encode the evolution of these multiplex interactions.

3) Temporal-Specific Models: This category of DTDG models employs GNN for processing graph structural data and non-RNN methods for temporal time series data processing.

Enhancing Accuracy. SGNN-GR [71] utilizes GAN [72] to generate synthetic time information, addressing catastrophic forgetting issues without additional data storage. Paired with GraphSAGE [6], it establishes a framework for DGNN. PI-GNN [26] employs parameter isolation for dynamic graphs to capture emerging patterns without compromising older ones. STGNP [73] introduces the spatio-temporal graph neural process, incorporating neural processes [74] for modeling spatio-temporal graphs. SILD [75] first explores distribution shifts in the spectral domain of dynamic graphs.

Improving Training Efficiency and Scalability. ROLAND [76] introduces embedding update modules to handle adjacent time snapshots, enabling the extension of static graphs to dynamic graphs. WinGNN [77] replaces time encoders with random sliding windows and adaptive gradients, effectively reducing the number of parameters. SpikeNet [31] replaces RNN with spiking neural networks (SNN) [65], reducing model and computational complexity through the leaky integrate-and-fire (LIF) model of SNN.

Specialized Applications. RT-GCN [21] predicts stocks using relationship and time convolution, employing GCN for relationship convolution and TCN for time convolution. RISD [78] is designed for heterogeneous graphs, utilizing heterogeneous GCN to learn graph representations and construct sampling graphs. STWave [79] is a framework for traffic prediction, incorporating GAT for spatial dimensions and a transformer for temporal dimensions to identify spatio-temporal correlations. VI-DGNN [80] presents a model tailored for trajectory prediction tasks in the intelligent transportation domain.

4) Comprehensive-Specific Models: The final category includes models that neither use GNN nor RNN for dynamic graph information processing.

Enhancing Accuracy. DySAT [81] computes node representations using joint self-attention across both structural neighborhoods and temporal dynamics. Compared to state-of-the-art recurrent methods for modeling graph evolution, dynamic self-attention is not only efficient but also consistently achieves superior performance. Dyngraph2vec [82] is an early Dynamic GNN model capable of capturing temporal dynamics. DynGEM [83] initializes snapshot embeddings from previous time steps before gradient training, avoiding learning from scratch. STADGNN [84] employs a spatiotemporal attention mechanism to dynamically capture local correlation shifts across sequences and temporal dependencies within sequences, eliminating the need for predefined priors.

Improving Scalability. T-SIRGN [85], inspired by SIRGN [86], extends the capabilities of existing approaches to handle efficiency and scalability issues in dynamic graphs.

Path-based Methods. The idea of DynamicTriad [87] is to impose triad [88] to model the dynamic changes of the network structure. Through this triadic closure process, DynamicTriad is able to capture the dynamic changes of the network and learn the representation vector of each vertex at different time steps. Netwalk [20] employs a deep auto-encoder model to reconstruct node representations and effectively group embeddings along intricate walking paths within the graph.

Capturing Global Information. DEFT [89] employs learning spectral wavelets [90] to capture global features, effectively learning dynamic evolution graph representations. DREAM [91] captures long-term evolution through a temporal self-attention network. EAGLE [92] utilizes the *modeling-inferring-discriminating-generalizing* paradigm to enhance extrapolation capabilities in the future.

Specialized Applications. STGCN [18] proposes a new deep learning framework to solve the problem of time series prediction in the transportation field. Instead of applying regular convolutions and recurrent units, it formulates the problem on the graph and builds a model with a full convolutional structure, making training faster and with fewer parameters. TEDIC [93] models information flow using enhanced graph convolution and extracts fine-grained patterns over time for dynamic social interaction pattern extraction. DEGC [19] addresses recommendation system challenges with an isolation-based approach to handle obsolescence. FiFrauD [94] is an unsupervised and scalable method for detecting suspicious traders and behavioral patterns efficiently. DyGNNExplainer [95] introduces a causality-inspired generative model based on structural causal models to explore the philosophy of DyGNN prediction by identifying trivial, static, and dynamic causal relationships.

5) Discussion: From Fig. 3 it is evident that the most common approach in DTDG models is the combined use of GNNs and RNNs to encode both structural and temporal information. Additionally, utilizing either GNN or RNN in conjunction with other specialized modeling techniques, such as temporal-specific and structural-specific methods, is another viable option. Finally, there are several models that do not utilize GNNs or RNNs but instead employ alternative modules like Self-Attention (SA) and Temporal Attention (TA) to encode structure and time.

While leveraging established methods to inform DTDG models remains prevalent, there are both advantages and disadvantages to this approach. In many cases, conventional GNN and RNN architectures prove effective in solving tasks and achieving high performance. However, the combination of GNN and RNN can result in complex models and lengthy training times, which may not be ideal for specific tasks or data characteristics. Some datasets may require more adaptable or customized architectures to effectively capture their unique attributes. For instance, in scenarios involving relational or heterogeneous graphs, standard GNNs may not suffice, necessitating novel graph structure learning approaches. RNNs can encounter issues like gradient vanishing or exploding gradients when processing long sequences, while also demanding significant computational resources. As an alternative, replacing RNNs with attention mechanisms represents a promising direction to address these challenges.

B. Continuous-Time Dynamic Graph Models

In the realm of CTDG models, we categorize them into instant node update model, neighbor aggregation-update model,

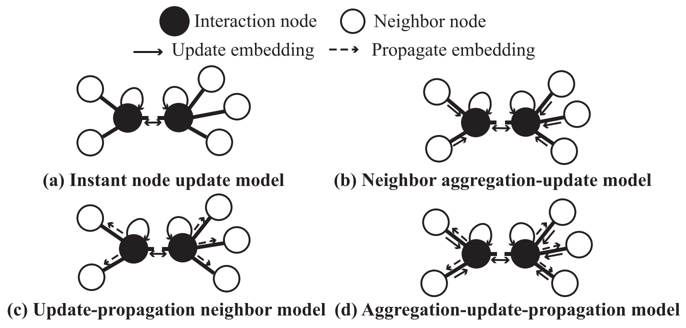
Interaction node
Neighbor node
Update embedding
Propagate embedding
(a) Instant node update model
(b) Neighbor aggregation-update model
(c) Update-propagation neighbor model
(d) Aggregation-update-propagation model

Fig. 4. Node update methods based on event streams.

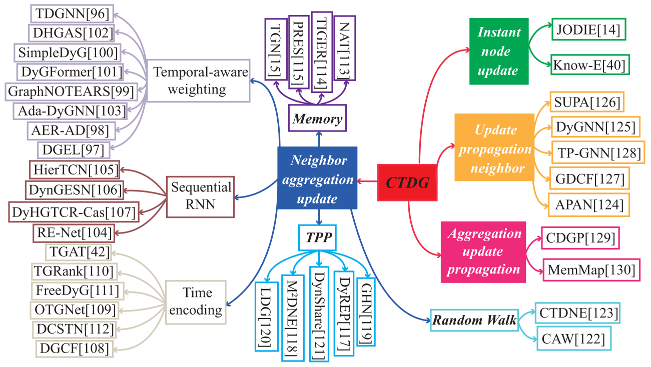

Fig. 5. The taxonomy for CTDG models in our survey.

update-propagation neighbor model, and aggregation-update-propagation model according to their methods for updating nodes using event streams. The distinctions between these four node update methods are depicted in Fig. 4. The taxonomy of CTDG models in our survey is shown in Fig. 5.

1) Instant Node Update Models: These models update the node embedding based solely on the current event and directly involved nodes, without considering the influence of neighbors, as depicted in Fig. 4(a). JODIE [14] is a coupled RNN model that uses two RNNs to update the embeddings of users and items for each interaction. Moreover, it introduces a new projection operator to model the future embedding trajectories of users/items. Know-E [40] is pioneering in employing TPP for CTDG and is capable of predicting potential event occurrence times. It updates node embeddings based on incoming events and recent relationships and time series.

2) Neighbor Aggregation-Update Models: These models leverage a combination of event data, historical node information, and neighbor information to update node embeddings, as shown in Fig. 4(b). This approach is frequently employed in CTDG models and represents a conventional method for dynamic graph embedding. Within this category, classification can be further refined based on the methods used to integrate time information for graph updates:

Temporal-Aware Weighting Models. TDGNN [96] introduces a dynamic network representation learning framework that effectively captures node representations by incorporating node characteristics and edge time information using the TDAgg aggregation function. DGEL [97] outlines three key processes (inherent interaction potential, time attenuation neighbor enhancement, and symbiotic local learning) to comprehensively update dynamic node embeddings with rich graph information. AER-AD [98] focuses on inductive anomaly detection in attribute and non-attribute dynamic graphs, utilizing anonymous edge representation for detecting anomalies in dynamic bipartite graphs in an inductive setting. GraphNOTEARS [99] first studies the learning problem of directed acyclic graphs and develop a score-based learning method. SimpleDyG [100] proposes a novel strategy that maps dynamic graphs to sequences to enhance scalability. DyGFormer [101] presents a transformer-based dynamic graph learning architecture that effectively captures node correlations and long-term temporal dependencies. DHGAS [102] introduces the first dynamic heterogeneous graph neural architecture search method, featuring a unified dynamic heterogeneous graph attention (DHGA) framework that allows each node to focus on its heterogeneity and dynamic neighbors simultaneously. Ada-DyGNN [103] introduces a robust knowledge adaptation framework for dynamic GNNS using reinforcement learning, enabling it to adaptively select which nodes to update.

Sequential RNN Models. RE-Net [104] models time, relationships, and interactions between nodes by utilizing RNN to capture the dynamics of time and relationships. Its neighborhood aggregation module combines information from neighboring nodes to handle multiple concurrent interactions at the same timestamp. HierTCN [105] employs RNN at the high-level to learn long-term interests, while TCN [47] is used at the low-level to predict the next interaction based on long-term interests and short-term interactions. DynGESN [106] introduces echo state networks (ESN) [45] for dynamic graph modeling. DyHGTCR-Cas [107] proposes a unified dynamic heterogeneous graph for information cascade prediction aimed at extracting uniform spatio-temporal features.

Time Encoding Models. Models in this category incorporate time information as part of the embedding to influence the calculation process. TGAT [42] is a standard temporal neighbor sampling method, which jointly encodes the hidden layer embeddings of neighbors and the time difference information, and similarly passes the information of nodes and neighbors through the transformer to obtain the final embedding for downstream tasks. DGCF [108] effectively models dynamic user-project relationships, capturing both collaborative and sequential connections. OTGNet [109] extends TGAT's application to open graphs, enabling the handling of open-time dynamic graphs. TGRank [110] boosts the model's link prediction expressiveness, allowing for the prediction of crucial structural information. FreeDyG [111] introduces a node interaction frequency encoding module that explicitly captures common neighbor proportions and node pair interaction frequencies to address the *shift* phenomenon. DCSTN [112] employs a cross-attention-based fusion mechanism, leveraging dynamic causal relationships to guide integration across time.

Memory Models. A part of CTDG models employs a memory mechanism to retain historical information for facilitating embedded updates. TGN [15] leverages memory to store historical node data, enabling updates upon the arrival of event streams. When the forward propagation is coming, the historical messages will be transformed into historical embeddings to participate in the operation. NAT [113] introduces a unique dictionary-based neighborhood representation and N-cache data structure to enable parallel access and updates of these dictionary representations on GPUs. TIGER [114] introduces a dual memory module to address the limitations of TGN. PRES [115] utilizes gaussian mixture models for correction and predicts new memory based on historical time series gradients. While this memory mechanism based CTDG approach inevitably leads to obsolescence issues, recent work [116] has computationally limited the information loss per training batch to maintain model effectiveness.

TPP Models. TPP is a valuable tool for modeling temporal dynamics, as discussed in Section II-C. Several works [117], [118], [119], [120], [121] utilize TPP in their frameworks. DyRep [117] views representation learning as a process that Bridges topology evolution and inter-node activity. It captures the interlacing dynamics of these two processes through a two-time scale deep time point process model, and uses a temporal attention mechanism to encode the structural information of time evolution into the node representation, thereby driving the nonlinear evolution of graph dynamics. M^{2}DNE [118] introduces macro constraints to scale the network through equations, complemented by micro-level time attention aggregation. GHN [119] utilizes the Hawkes process to capture entity interaction time dynamics in TKGs and introduces a novel time knowledge graph connection prediction ranking metric. LDG [120] enhances previous approaches by addressing issues related to long-term edge information quality. DynShare [121] designs an interval-aware personalized projection operator using TPP to enable diverse user mode selections within the same time interval. While these algorithms all leverage TPP, variations exist in their implementations. For instance, DyREP, GHN, LDG, and DynShare directly incorporate TPP in the entire forward propagation calculation, whereas M^{2}DNE use TPP for objective function computations.

Random-Walk-Based Models. Random walk has proven to be effective in graph learning, as discussed in Section II-C. CAW [122] is a technique for representing network dynamics, which automatically extracts temporal motifs through a temporal random walk, avoiding the complex selection and counting process in traditional methods. It employs an anonymization strategy that replaces node identities with hit counts to maintain inductive bias and establish inter-topic correlation. CTDNE [123] is a pioneering algorithm in dynamic graph embedding that incorporates time embeddings into network embeddings. This model introduces a comprehensive framework that integrates time dependencies into node embeddings and employs a depth map model through random walks.

3) Update-Propagation Neighbor Models: These models only utilize event and historical node information to update, embed, and propagate data to neighbors, as depicted in Fig. 4(c). APAN [124] designs a special mailbox module to store the information of historical neighbors, and adopts the attention mechanism of Transformer [52] to aggregate the information of neighbors. After that, the batch of interactive information is transmitted to the neighbor's mailbox in an asynchronous way to update the mailbox. DyGNN [125] can model dynamic information as the evolution of a graph. It continuously updates node information by coherently capturing the order information of edges (interactions), the time interval between edges and information propagation. SUPA [126] generates a sample graph for the event-involved nodes, updates them with event flow data, and propagates the updated information throughout the sample graph. GDCF [127] focuses on crowd flow modeling and employs a memory module to enhance update speed. TP-GNN [128] captures long-term dependencies through an innovative messaging approach that is based on the information flow between nodes.

4) Aggregation-Update-Propagation Models: Models in this category rely on historical neighbor information for embedding updates, followed by propagating the updated information to neighbors. CDGP [129] and MemMap [130] are examples of models in this category. CDGP focuses on popularity prediction in community dynamic graphs by identifying the community for the event, aggregating nodes within the community, and extending the community's influence to other nodes. MemMap introduces a memory structure called Memory Map to capture node correlations within a grid and aggregates and propagates information based on these correlations. This model type is less common because, typically, aggregating or propagating information from neighbors alone is adequate for learning or acquiring information. Models in this category perform both aggregation and propagation, which can lead to redundant computation. The mentioned methods utilize these processes for various functions in specific applications; for instance, CDGP [129] uses aggregation to capture community intensity and propagation to represent social influence within the community.

5) Discussion: Fig. 5 illustrates that most CTDG models belong to the neighbor aggregation-update category, which captures how historical event neighbors influence new events in a fresh event stream. However, there are situations where neighbors have minimal impact on new events, making the instant node update model more appropriate. Alternatively, when considering the reciprocal influence between events and their neighbors, the update-propagation neighbor model may be a more effective solution. In cases where events and neighbors mutually influence each other, the aggregation-update-propagation model is recommended.

For researchers focusing on CTDG models, there is a clear need to explore alternative methods beyond aggregating neighbors, presenting an avenue for further investigation. Similarly, for developers working on DGNNs, constructing the overall architecture based on neighbor aggregation and propagation processes can be a fruitful approach. For instance, integrating a memory component into the neighbor aggregation module and a mailbox component into the neighbor propagation module could enhance the model's capacity to capture and propagate relevant information effectively.

IV. Dynamic GNN Training Frameworks

In this section, we offer a survey of the latest dynamic GNN frameworks, encompassing 7 DTDG and 10 CTDG frameworks.

We start by outlining the requirements for dynamic GNN frameworks, then delve into the existing DTDG frameworks and CTDG frameworks, respectively.

A. Needs for DGNN Frameworks.

Despite significant progress in dynamic GNNs, their practical use is limited by challenges in training scalability and computational efficiency. Current models excel in accuracy and application diversity but struggle with scalability on large, evolving graphs due to their complexity. This calls for frameworks that optimize system execution, particularly in distributed training, but several challenges arise. First, dynamic load balancing is complex due to the interconnected nature of graph data, requiring continuous rebalancing different batches of subgraphs across trainers to avoid underutilization. Second, DGNN-specific temporal modules, like memory states and mailboxes, increase communication costs in distributed settings, adding to the already high overhead of static GNN training. Third, chronological dependencies in DGNNs limit parallelization, leading to sequential processing of timesteps and underutilization of hardware resources. Finally, the rise of heterogeneous systems (e.g., CPUs, GPUs, FPGAs) adds complexity, as coordinating workloads across diverse types of accelerators exacerbates load imbalance, communication bottlenecks, and synchronization challenges. Collectively, these challenges highlight the need for system-level innovations that optimize distributed execution, reduce communication, handle temporal constraints, and adapt to hardware diversity to facilitate efficient DGNN training on large dynamic graphs.

Recently, specialized GNN training frameworks like PyG [147] and DGL [148] have been developed to simplify programming and enhance training efficiency for various GNN models. Many optimizations are proposed on top of fundamental training modules. For instance, NeuGraph [149] and Roc [150] optimize vertex access and load balancing through improved graph partitioning. AliGraph [151] and DUCATI [152] reduce data transmission by caching embeddings and intermediate training results. P3 [153] and PipeGCN [154] employ accelerated parallel training. Optimizations are also made in heterogeneous systems. NeutronOrch [155] improves GNN training in CPU-GPU environments through hierarchical task scheduling. GraphACT [156] optimizes GCN training on a CPU-FPGA platform by reducing data communication overhead. GALA-GNN [157] uses a heuristic graph partitioning algorithm to address missing neighborhoods in subgraphs, optimize task allocation, and improve resource utilization on heterogeneous platforms. However, these frameworks primarily cater to training static graphs and lack essential mechanisms for updating graphs and modeling temporal information in dynamic GNNs. Key modules like dynamic graph loading, temporal neighbor sampling, and temporal message passing are missing. The temporal dependency of dynamic graphs also complicates parallel computing. Therefore, users must develop these modules themselves to construct dynamic GNN models and carefully handle dynamic graphs to ensure the temporal consistency, resulting in heavy development costs for users.

In the past two years, several dynamic GNN training frameworks have emerged, aiming to better support dynamic GNN training, including both discrete-time and continuous-time dynamic graph models. However, due to variations in training processes between these model types, most frameworks are tailored for either one. We conducted a study on the existing 17 dynamic GNN training frameworks and compiled a detailed comparison of their universality, expandability, and supported functionalities, as outlined in Table II.

B. Discrete-Time DGNN Frameworks

Discrete-time DGNNs typically treat the dynamic graph as a sequence of graph snapshots. PyGT [131], an extension of PyG, introduces three snapshot-based dynamic graph types and provides a unified data loader for each type of dynamic graph to support training DTDG models. However, it is limited to full-batch training on a single GPU for small-scale graphs. ESDG [132] utilizes snapshot partitioning to distribute multiple graph snapshots to different devices for parallel GNN training, and after finishing the GNN computation, redistributes them to colocate the same vertices from different snapshots on the same device for parallel RNN computation. PiPAD [133], based on the similarity of the structures between adjacent graph snapshots, reduces data transmission by transmitting the common parts of a set of consecutive graph snapshots and the individual parts of each graph snapshot, ensuring that these snapshots can be computed in parallel on GPU. DynaGraph [134] proposes a time fusion mechanism to concatenate node features of multiple snapshots in a single graph convolution operation to fully utilize the GPU resource. BLAD [135] explores fine-grained operator-level parallelism to fully leverage GPU resources by simultaneously executing memory-intensive graph operators and compute-intensive neural network operators on a single GPU. DGC [136], unlike other frameworks, divides dynamic graphs into chunks for distributed training. It employs a chunk generation algorithm based on graph coarsening, considering spatio-temporal non-uniformity to balance training workload and reduce communication costs, improving DGNN training efficiency. STGraph [137] seamlessly integrates with dynamic graph data structures for enhanced DTDG processing. By constructing snapshots on demand during training, it reduces memory overhead while maintaining efficiency.

C. Continuous-Time DGNN Frameworks

Traditional static graph storage formats like compressed sparse row (CSR) struggle with efficient handling of edge and vertex insertions and deletions in dynamic graphs. Dynamic graph storage also needs to handle temporal graph sampling effectively, considering only edges up to the current timestamp. TGL [138] addresses these challenges by introducing a framework with components such as temporal sampler, temporal message mailbox, vertex history memory, memory updater, and message passing engine. It utilizes the T-CSR format for dynamic graph storage, along with parallel samplers and random block scheduling techniques to enhance training efficiency. GNNFlow [144] introduces a time-indexed block-based data

TABLE II
THE COMPARISON BETWEEN THE EXISTING DYNAMIC GNN TRAINING FRAMEWORKS. "SMSG" MEANS "SINGLE-MACHINE SINGLE-GPU", "SMMG" MEANS "SINGLE-MACHINE MULTI-GPUS", "MMMG" MEANS "MULTI-MACHINES MULTI-GPUS"

Features / Systems	Universality		Hardware Support			Supported functionalities			
	DTDG	CTDG	SMSG	SMMG	MMMG	Temporal neighbor storage	Feature extraction optimizing	Temporal parallel training	Open-sources
PyGT [131]	✓	✗	✓	✗	✗	Store snapshots	✗	✗	✓
ESDG [132]	✓	✗	✓	✓	✓	Store snapshots	✗	Snapshot parallelism	✗
PiPAD [133]	✓	✗	✓	✓	✗	Store snapshots	Extract common snapshots	Snapshot parallelism	✗
DynaGraph [134]	✓	✗	✓	✓	✓	Store snapshots	✗	Snapshot partition parallel	✗
BLAD [135]	✓	✗	✓	✓	✓	Store snapshots	✗	Operator parallelism	✗
DGC [136]	✓	✗	✓	✓	✓	Store chunks	Vertex embeddings cache	Subgraphs partition parallel	✗
STGraph [137]	✓	✗	✓	✗	✗	Store snapshots	State stack	✗	✓
TGL [138]	✓	✓	✓	✓	✗	Store sorted neighbors	✗	Small-batch parallelism	✓
DistTGL [139]	✓	✓	✓	✓	✓	Store sorted neighbors	✗	Memory parallelism	✓
DisTGL [140]	✓	✓	✓	✓	✓	Store event stream	Static feature cache	Two-stage parallel fetch	✗
TGLite [141]	✓	✓	✓	✗	✗	Store sorted neighbors	Deduplication optimization	Temporal neighbor sampling	✓
SPEED [142]	✗	✓	✓	✓	✗	Store event stream	✗	Subgraph partition parallel	✓
NeutronStream [143]	✗	✓	✓	✗	✗	Store event stream	✗	Event group parallelism	✓
DyGLib [101]	✗	✓	✓	✗	✗	Store event stream	✗	✗	✓
GNNFlow [144]	✗	✓	✓	✓	✓	Store event stream	Vectorized cache	✗	✓
MSPipe [145]	✗	✓	✓	✓	✓	Store event stream	✗	Staleness-aware pipeline	✓
Sven [146]	✗	✓	✓	✓	✓	Store event stream	✗	Hierarchical pipeline Parallelism	✗

structure for dynamic graph storage to facilitate edge sampling and updates. It also implements caching mechanisms to optimize CPU-GPU data transfer performance by caching frequently used edge features. TGLite [141] proposes a lightweight programming framework that integrates optimizations of TGL for temporal graphs, and designs cache, preload, and TGOpt [158] based Deduplication to accelerate training. Its greatest contribution lies in providing user-friendly programming. CTDG models view dynamic graphs as sequences of events with temporal dependencies, making parallelization challenging. TGL uses mini-batch parallelism but overlooks dependencies within individual mini-batches, potentially impacting training accuracy. NeutronStream [143] employs an adaptive sliding window training approach to capture temporal dependencies and identify parallelizable event groups while maintaining temporal order. DistTGL [139] extends TGL by introducing distributed training methods, such as epoch parallelism and memory parallelism, to enhance scalability across multiple GPUs. DisTGL [140] utilizes a time-aware partitioning scheme and a range of enhanced communication technologies to ensure efficient distributed computing and minimize communication overhead. SPEED [142] customizes partitioning strategies for parallel training to ensure load balance and minimize replication factors while preserving temporal information. DyGLib [101] offers a library for dynamic graph learning, featuring standardized training processes, scalable coding interfaces, and thorough evaluation protocols. MSPipe [145] is a versatile and efficient memory-based TGNN framework that maximizes training throughput while maintaining model accuracy. Sven [146] is a library of co-designed algorithms designed to accelerate TGNN training on multi-GPU platforms.

V. DYNAMIC GNN BENCHMARKS

A. Datasets

Table III shows a summary of commonly used datasets in dynamic GNN evaluation. It includes details on application domains, node and edge counts and dimensions, timestamp ranges, label information, and dataset sizes. The datasets selected for evaluation are highlighted in bold (Section VI).

TABLE III
SUMMARY OF COMMON GRAPH DATASETS, WHERE $|V|$ AND $|E|$ STAND FOR THE NUMBER OF NODES AND EDGES, $Range(t)$ DENOTES THE TIMESTAMP RANGE, AND $Sizes$ SPECIFIES THE TOTAL FILE SIZES

Datasets	$\|V\|$	$\|E\|$	$Range(t)$	$Sizes$
Reddit [159]	10984	672447	0∼2678390	2.2GB
DGraphFin [160]	4889537	4300999	1∼821	649MB
Enron [161]	184	125,235	0∼113740399	35MB
Facebook [162]	63731	1269502	1157454929∼1232576125	26MB
Social Evolution [163]	74	2,099,520	1188972131∼1247740843	148MB
UCI [164]	899	33720	1084560796∼1098772901	668KB
Wikipedia [159]	9227	157474	0∼2678373	534MB
MOOC [159]	7144	411749	0∼2572086	40MB
ML25M [165]	221588	25000095	789652009∼1574327703	647MB
LastFM [159]	1980	1293103	0∼137107267	37MB
FB-FORUM [166]	899	33720	1084585996∼1098798101	612KB
DBLP [167]	28,086	162,451	1∼27	375MB
Yelp [168]	2138275	6990280	1108495402∼1642592925	5.0GB
ICEWS05-15 [169]	10,094	461,329	1104537600∼1451520000	30MB
ICEWS14 [169]	6,869	96,730	1388534400∼1419984000	6MB
ICEWS18 [170]	23,033	741820	1514764800∼1546214400	184MB
GDELT [171]	16682	191290882	0∼175283	82.3GB
Bitcoin-OTC [172]	5881	35592	1289241942∼1453684324	988KB
Bitcoin-Alpha [172]	3782	24186	1289192400∼1453684324	492KB
AS-733 [172]	7716	11965533	939340800∼946771200	115MB
Flights [161]	13169	1927145	0∼121	32MB

*1) **Social Networks:*** **Reddit** [14] consists of one month of user posts on subreddits, with users and subreddits as nodes and timestamped posting requests as links. **DGraphFin** [37] is the real social network in the financial industry provided by Finvolution Group, with nodes representing Finvolution users and edges indicating emergency contacts. **Enron** [173] is a mail dataset capturing interactions among Enron Inc. employees, where communication links denote email exchanges among core employees. **Facebook** [174] is a user interaction network where nodes represent users and edges signify interactions between users. **Social Evolution** [175] comes from MIT Human Dynamics Lab, focusing on social relationship evolution and the degree of closeness between individuals. **UCI** [176] is an online community of students from UC Irvine, with links representing messages exchanged between users.

2) Interaction Networks: **Wikipedia** [14] is a bipartite interaction graph capturing user edits on Wikipedia pages over a month, with nodes representing users and pages, and links representing editing behaviors. **MOOC** [14] records student behaviors in Massive Open Online Courses (MOOCs), such as watching videos and submitting answers. **ML25 M** [177] contains user ratings of movies, reflecting their preferences and levels of interest in various movies. **LastFM** [14] provides information on user-listened songs over a month. **FB-FORUM** [178]is the Facebook-like forum network from the same online community as the social network dataset, focusing on user activities in the forum. **DBLP** [179] captures an academic collaboration network dataset with nodes representing authors and edges denoting co-authored papers. **Yelp** [81] is a business review website where users can review businesses and discover interested ones based on others' comments.

3) Event Networks: **ICEWS** [180] and **GDELT** [181] are event networks where nodes represent actors and edges represent point-time events derived from news and articles, such as *yield, threaten and make public statements*, providing a dynamic view of real-world interactions and occurrences.

4) Trade Networks: **Bitcoin-OTC** [182] and **Bitcoin-Alpha** [183] are who-trust-whom networks of users trading on the bitcoin-otc platform and bitcoin-alpha platform.

5) Traffic Networks: **AS-733** [184] is a communication network composed of routers used to exchange traffic with peers. **Flights** [185] is a network where nodes represent airports and edges represent flights between these airports.

TABLE IV
COMMON EVALUATION METRICS IN DYNAMIC GNNS

Applications	Metrics	Equations
Binary classification	Precision	$\frac{TP}{TP+FP}$
	Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$
	Recall	$\frac{TP}{TP+FN}$
	F1 score	$\frac{2\times Precision\times Recall}{Precision+Recall}$
	AUC	$TPR=\frac{TP}{TP+FN}, FPR=\frac{FP}{FP+TN}$ AUC is the area under the TPR-FPR curve
	AP	AP is the area under the Precision-Recall curve
Multi classification	Micro-F1	$(1) Precision_{micro}=\frac{\sum_i TP_i}{\sum_i TP_i+FP_i}$ $(2) Recall_{micro}=\frac{\sum_i TP_i}{\sum_i TP_i+FN_i}$ $\frac{2\times Precision_{micro}\times Recall_{micro}}{Precision_{micro}+Recall_{micro}}$
	Macro-F1	$(1) Precison_{macro}=\frac{1}{N}\sum_i \frac{TP_i}{TP_i+FP_i}$ $(2) Recall_{macro}=\frac{1}{N}\sum_i \frac{TP_i}{TP_i+FN_i}$ $\frac{2\times Precison_{macro}\times Recall_{macro}}{Precison_{macro}+Recall_{macro}}$
Recommendation system	Precision@K	$\frac{TP@K}{TP@k+FP@K}$
	Recall@K	$\frac{TP@K}{TP@k+FN@K}$
	MR	$\frac{1}{\lvert S\rvert}\sum_{i=1}^{\lvert S\rvert} rank_i$
	MRR	$\frac{1}{\lvert S\rvert}\sum_{i=1}^{\lvert S\rvert} \frac{1}{rank_i}$
	HITS@K	$\frac{1}{\lvert S\rvert}\sum_{i=1}^{\lvert S\rvert} IF(rank_i \leq k)$
Regression Model Performance	RMSE	$\sqrt{\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{n}}$
	MAE	$\frac{\sum_{i=1}^{n}\lvert y_i-\hat{y}_i\rvert}{n}$
	MAPE	$\frac{\sum_{i=1}^{n}\lvert\frac{y_i-\hat{y}_i}{y_i}\rvert}{n}$

TABLE V
CONFUSION MATRIX IN BINARY CLASSIFICATION TASKS

		Prediction	
		Positive	Negative
Actual	True	True Positive(TP)	False Negative(FN)
	False	False Positive(FP)	True Negative(TN)

B. Metrics

Here is an overview of common evaluation metrics to assess the performance of various dynamic GNN models and frameworks. Table IV shows the detailed calculation equations.

1) Binary Classification Performance: The confusion matrix for binary classification tasks is shown in Table V, where rows represent predicted classes and columns represent actual classes. **Precision** measures the proportion of correctly predicted positive samples among all predicted positives. **Accuracy** indicates the proportion of correctly predicted samples out of all samples. **Recall** assesses the proportion of actual positive samples correctly predicted. **F1 Score** is the harmonic mean of precision and recall, offering a balanced measure of classification performance for both positive and negative cases. **AUC** (Area under the curve) typically denotes the area under the ROC curve, a tool for assessing binary classification models using True Positive Rate (TPR) and False Positive Rate (FPR). **AP** (Average Precision) is a common metric used to evaluate the performance of classifiers or retrieval systems. AP is the area under the Precision-Recall curve.

2) Multi Classification Performance: In multi-class classification tasks, key performance metrics include: **Micro-F1 Score** computes the F1 score considering total true positives, false positives, and false negatives across all categories. **Macro-F1 Score** computes the average of F1 scores for individual categories. **Micro-AUC** is calculated by micro-averaging true positive rates and false positive rates for all categories. **Macro-AUC** is obtained by macro-averaging AUC values for each category.

3) Recommendation System Performance: In recommendation system tasks, **Precision@K** measures the proportion of the top K recommended items to the total number of recommended items. **Recall@K** measures the proportion of the top k recommended items to the total number of relevant items. **MR** (Mean Rank) measures the average recommended ranking $rank_i$ of all users S. **MRR** (Mean Reciprocal Rank) measures the average reciprocal of the recommended ranking of all users. **HITS@K** measures the accuracy of the top K recommendation results by assessing how many of them align with users' genuine interests. Function IF$(\cdot)$ outputs 1 if the condition is true and 0 otherwise.

4) Regression Model Performance: In recommendation systems, **RMSE** (Root Mean Squared Error) quantifies the difference between predicted and actual values, while **MAE** (Mean Absolute Error) calculates the average error between predicted and actual values. **MAPE** (Mean Absolute Percentage Error) measures the average percentage difference between predicted and actual values.

5) Efficiency and Scalability: **Throughput** measures the number of tasks processed within a specific time frame. In deep learning, it typically refers to the number of samples that the model can handle per unit of time, providing insights into processing speed. **Training time** refers to the duration time the model spends in the training phase. **GPU memory usage** indicates the amount of memory occupied during training or inference on a GPU. Efficient memory management is crucial for optimizing model performance and scalability. **Parameter size** quantifies the total number of learnable parameters in the model. The parameter size directly impacts the model's complexity, storage requirements, and computational efficiency, influencing scalability and performance.

VI. Experimental Comparison

We thoroughly evaluate various dynamic GNN models and frameworks in terms of training accuracy, efficiency, and memory usage. We first compare several classic DTDG and CTDG GNN models (Section VI-B), then compare models implemented on optimized frameworks (Section VI-C), and analyze multi-GPU scalability (Section VI-D). In addition, we explore the impact of hyperparameter settings on performance and efficiency(Section VI-E), and finally discuss the node classification task(Section VI-F).

A. Experiment Setting

Test Setup: We perform our experiments on two Ubuntu 22.04.3 LTS machines, each featuring an Intel Xeon Gold 6342 CPU@2.80 GHz and four Nvidia A40 48 GB GPUs, equipped with 1 TB of memory and 40 TB of disk space.

Hyperparameter setting: The default configuration for all CTDG models includes one GNN layer and a batch size of 1000, which is a commonly used setting in dynamic GNN. Additionally, to ensure consistency, the models adhere to the same early stopping condition (3 epochs) as AP. For DTDG models, uniform snapshot intervals are maintained across datasets. The standard train/validation/test dataset split of 70%/15%/15% is followed. Remaining parameters are set to default values. These settings are widely adopted and generally yield optimal performance. We also evaluate performance on different hyperparameter settings in Section VI-E.

Baseline Models and Frameworks: We compare seven CTDG GNN models (JODIE, TGAT, TGN, APAN, CAW, DyREP, DyGFormer) and three DTDG GNN models (EvolveGCN, Roland, DySAT), along with five dynamic GNN frameworks (TGL, DistTGL, SPEED, DyGLib, TGLite). For JODIE and DyREP models, we use the implementation provided by TGN. We utilize the most efficient version, TGN-attn, for the TGN model. EvolveGCN has two versions EvolveGCN-H and EvolveGCN-O, using GRU and LSTM as components of learning time, respectively.

Datasets and Metrics: We select six graph datasets for evaluation based on their common usage in existing works. The datasets include Wikipedia, Reddit, MOOC, Flights, ML25 M, and DGraphFin. Among these, Wikipedia, Reddit, and MOOC

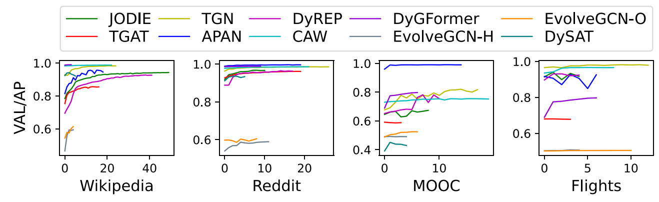

Fig. 6. Val.ap after each training epoch.

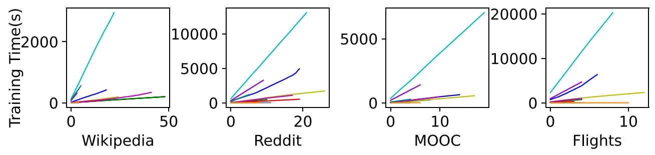

Fig. 7. Total training time cost after each epoch.

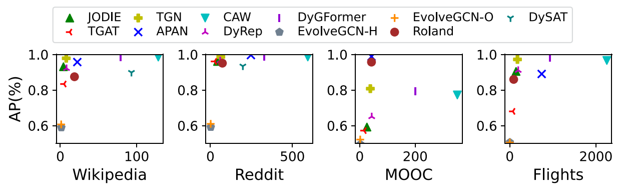

Fig. 8. The AP(%) metric and average training time(s) per epoch.

are most frequently utilized, while Flights, DGraphFin, and ML25 M are larger datasets chosen specifically to assess the scalability of dynamic GNN models and frameworks. Additional details about the datasets can be found in Table III. For Wikipedia, Reddit, ML25 M, and DGraphFin, we use the default feature dimension. For MOOC and Flights, we initialize the feature size to 172 dimensions. In the case of DTDG methods, we employ different time intervals for each dataset: 100,000 for Wikipedia, Reddit, and MOOC; 7 for Flights; 50 for DGraphFin; and 32,000,000 for ML25 M.

When evaluating training accuracy, we focus on four key metrics (AUC, AP, Recall, and Accuracy) that are commonly used by the selected models and frameworks. Additionally, we analyze time cost and memory usage to evaluate training efficiency and scalability, which are essential indicators of a framework's performance. Our main emphasis is on the link prediction task, the most prevalent task for dynamic GNNs. We also assess the node classification task in Section VIII.

B. Comparison of Dynamic GNN Models

We conducted an extensive comparison of ten dynamic GNN models by training them to convergence, with the accuracy results detailed in Table VI. Most models encountered out-of-memory (OOM) issues on the large DGraphFin and ML25 M datasets, except for APAN and CAW. Even APAN and CAW exhibited significant slowness and failed to complete within 48 hours, so the results on these two datasets are excluded. Furthermore, we illustrate the convergence curves of validated AP, the time cost after each epoch, the relationship between the average training time and AP, and the GPU memory usage in Figs. 6, 7 8, and 9 respectively. Note that, Roland's distinctive training method allows it to converge after just one epoch, hence its results are not included in Figs. 6 and 7.

TABLE VI
CONVERGE ACCURACY(%) ON DYNAMIC GNN MODELS FOR LINK PREDICTION TASK

Type	Model	Wikipedia				Reddit				MOOC				Flights			
		AUC	AP	Recall	Accuracy	AUC	AP	Recall	Accuracy	AUC	AP	Recall	Accuracy	AUC	AP	Recall	Accuracy
CTDG	JODIE	93.25	93.13	86.57	85.15	96.36	96.10	91.55	90.19	62.43	59.20	68.05	59.42	91.16	90.49	93.95	78.62
	TGAT	82.02	83.39	78.38	73.51	95.85	96.10	91.51	88.77	56.37	57.20	84.19	54.60	70.08	68.00	65.89	63.73
	TGN-attn	97.81	97.88	90.76	91.91	98.61	98.62	94.50	94.10	82.88	80.78	67.15	74.03	97.63	97.22	98.33	89.27
	APAN	96.50	95.73	**95.79**	90.94	**99.69**	**99.65**	**98.86**	**97.77**	**99.35**	**98.83**	**98.96**	**98.59**	89.95	89.04	94.11	78.18
	CAW	98.19	**98.54**	91.51	**94.30**	98.35	98.59	92.10	93.93	74.24	77.24	87.99	62.18	96.26	96.63	88.43	90.53
	DyRep	91.71	92.14	81.95	82.93	96.67	96.54	89.34	90.67	69.97	65.05	43.25	60.07	92.14	91.09	**98.42**	70.94
	DyGFormer	**98.29**	98.53	91.71	93.49	98.69	98.88	92.54	95.31	78.23	79.21	62.56	70.17	**98.01**	**98.15**	92.92	**93.84**
DTDG	EvolveGCN-H	56.96	58.91	38.25	56.04	58.31	59.13	56.28	55.93	49.26	49.51	27.74	49.65	50.36	50.59	89.08	50.06
	EvolveGCN-O	57.19	60.67	37.63	55.88	58.74	60.99	84.78	52.78	50.88	52.15	10.01	51.04	50.35	50.47	63.83	50.16
	Roland	85.84	87.48	74.73	81.45	93.29	95.11	78.94	84.54	94.09	95.71	79.81	75.82	85.12	85.99	36.40	51.68
	DySAT	86.68	89.87	92.93	65.13	91.79	93.45	94.71	71.95	45.51	43.36	55.96	48.62	OOM			

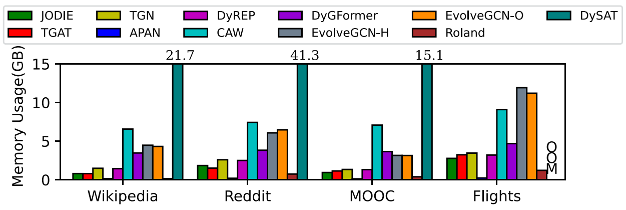

Fig. 9. Memory usage of training dynamic GNN models.

Table VI highlights the consistent top performance of CTDG models on various datasets and metrics. CAW stands out on the Wikipedia dataset, showcasing superior performance across all metrics due to its unique anonymous causal walk approach. APAN shines on the Reddit and MOOC datasets, demonstrating versatility beyond financial scenarios into user interaction datasets. DyGFormer excels in AUC/AP/Accuracy on the Flights dataset. The gap between the TGAT accuracy reported in our experiments and the results on the original paper is due to us using only one layer of the TGAT network. We report the results of the two-layer TGAT in Section VI-E. Figs. 7 and 9 show that the high training accuracy of CAW, APAN, and DyGFormer comes with significant time and memory costs, especially for CAW. TGN maintains commendable performance with acceptable training times. DySAT and EvolveGCN exhibit lower accuracy metrics while consuming considerable memory, making them less efficient. Fig. 6 indicates that higher validation metrics usually lead to superior test results. From Fig. 8, we know that the TGAT model of one layer, although the performance of the AP metric is suboptimal, consumes less time. Although CAW always achieves better performance, TGN and APAN, which can achieve relatively high accuracy with less time, would be preferred for researchers. In summary, the CTDG models often outperform the DTDG models. This is because the CTDG models are elaborately designed with a structure tailored for dynamic graphs, whereas the DTDG models merely stack GNN and RNN modules.

C. Comparison of Models on Frameworks

In this section, we evaluate these models trained on dedicated optimized dynamic GNN frameworks with open-source code, including TGL, DistTGL, SPEED, DyGLib, and TGLite. TGL implements JODIE, TGAT, TGN, APAN, and DySAT models, while DistTGL only focuses on the TGN model. SPEED covers TGN, TGAT, JODIE, DyREP, and TIGE models. For DyGLib, we use their implementations of JODIE, DyRep, TGAT, TGN, and CAW for comparison with the model. TGLite uses the already given JODIE, APAN, TGAT, TGN models. For TGL's DySAT implementation (TGL-DySAT), we maintain the same time interval size as the model for each dataset. We set $top_k = 10$ for SPEED to explore potentially higher evaluation metrics.

Accuracy Metrics Comparison: We show the three accuracy metrics of models trained to converge on frameworks in Table VII. Situations facing issues such as out-of-memory (OOM), or overtime are indicated in the corresponding bar. In general, frameworks tend to achieve better accuracy than the origin models in most cases, because of the system optimizations used in these frameworks. DyGLib doesn't provide any performance improvements on the models. This is due to the fact that it provides a unified library without any better optimizations. Frameworks have an advantage over models in their ability to handle large datasets effectively. Models often struggle with training on large datasets, whereas frameworks are tailored to support such scenarios efficiently. Among TGL, SPEED, DistTGL, DyGLib, and TGLite comparisons, TGLite frequently demonstrates higher accuracy. In particular, the TGN version of TGLite achieves the vast majority of optimal and suboptimal performance.

Training Time Comparison: The training times of these models to reach convergence on various frameworks were compared, with results displayed in Fig. 10. It indicates that frameworks generally require less training time compared to the origin models, DyGLib is excluded, for the same reasons mentioned above. This trend is especially prominent in TGL, where it consistently outperforms the original on most datasets and models. DistTGL, as a distributed version of TGL, despite having lower evaluation metrics than TGL, shows a reduction in training time. SPEED exhibits a significantly longer training time, with up to 10 times longer than TGL, although this disparity is due to the $top_k = 10$ setting in SPEED. DyGLib only implements a unified benchmark, without much improvement in efficiency. TGLite, as a recent work, has further performance and efficiency improvements over TGL and is one of the best frameworks to date. *GPU Memory Usage Comparison:* The GPU memory usage during the training of these models on various frameworks was compared, with results displayed in Fig. 11. Generally, the origin models tend to consume more memory compared to frameworks, indicating that frameworks are optimized for dynamic GNN training. However, in ML25 M and DGraphFin

TABLE VII
COMPARISON OF ACCURACY METRICS(%) OF TRAINING MODELS TO CONVERGE ON FRAMEWORKS. "ACC" MEANS "ACCURACY", "OOM" MEANS "OUT OF MEMORY". WE PRESENT THE OPTIMAL AND SUBOPTIMAL RESULTS IN BOLD AND UNDERLINED

Model	Version	Wikipedia			Reddit			MOOC			Flights			ML25M			DGraphFin		
		AUC	AP	Acc	AUC	AP	Acc	AUC	AP	Acc	AUC	AP	Acc	AUC	AP	Acc	AUC	AP	Acc
JODIE	-Origin	93.25	93.13	85.15	96.36	96.10	90.19	62.43	59.20	59.42	91.16	90.49	78.62	OOM			OOM		
	-TGL	98.12	97.63	94.34	99.43	99.27	97.27	99.26	98.66	99.04	90.29	89.01	74.00	98.64	98.39	94.83	74.88	75.14	69.73
	-SPEED	90.66	90.47	81.75	97.38	97.29	91.59	71.81	66.99	64.93	89.05	87.22	67.26	96.48	95.94	91.15	OOM		
	-DyGLib	91.44	91.56	83.30	97.34	97.33	91.33	78.54	73.69	73.05	90.96	89.38	77.94	OOM			OOM		
	-TGLite	97.80	97.19	93.98	99.63	99.55	97.66	99.39	98.98	99.03	93.07	92.35	84.60	98.54	98.25	94.51	**78.94**	**81.38**	**73.53**
TGAT	-Origin	82.02	83.39	73.51	95.85	96.10	88.77	56.37	57.20	54.60	70.08	68.00	63.73	72.31	68.92	66.33	57.76	55.94	55.18
	-TGL	88.56	85.77	81.21	95.74	95.62	88.32	62.91	59.73	58.41	71.23	65.46	67.10	63.09	62.95	58.87	57.28	54.75	56.59
	-SPEED	74.95	76.67	67.52	94.71	94.87	87.00	66.66	63.38	62.47	80.49	79.41	72.49	89.44	86.55	83.05	OOM		
	-DyGLib	65.40	70.55	59.89	93.38	93.46	85.43	56.23	54.24	52.63	73.42	68.69	67.74	89.32	86.39	83.22	71.43	75.03	66.53
	-TGLite	92.96	91.16	86.51	**99.76**	**99.73**	98.04	95.27	93.88	85.87	97.55	**97.41**	**92.38**	93.61	91.97	86.71	74.46	77.01	70.52
TGN	-Origin	97.81	97.88	91.91	98.61	98.62	94.10	82.88	80.78	74.03	**97.63**	97.22	89.27	OOM			OOM		
	-TGL	98.16	98.70	94.99	99.67	99.59	97.64	99.51	99.13	**99.32**	92.69	92.41	78.31	97.55	97.10	92.18	78.77	80.40	73.01
	-SPEED	98.04	98.16	92.62	96.44	96.42	87.46	86.98	84.73	79.00	93.51	92.95	69.03	95.66	94.78	87.08	OOM		
	-DistTGL	96.79	97.19	88.87	97.61	97.66	87.48	84.78	82.73	67.19	96.83	96.24	90.24	98.53	98.32	88.97	72.62	72.38	56.76
	-DyGLib	97.06	97.32	90.88	98.33	98.33	93.54	60.62	54.41	60.06	97.36	96.78	90.62	OOM			OOM		
	-TGLite	**99.12**	**98.82**	**96.50**	**99.76**	99.69	**98.16**	**99.54**	**99.23**	99.13	96.80	96.51	91.02	**98.74**	**98.61**	**95.07**	70.57	67.80	66.75
APAN	-Origin	96.50	95.73	90.94	99.69	99.65	97.77	99.35	98.83	98.59	89.95	89.04	78.18	Overtime			Overtime		
	-TGL	97.29	95.95	94.12	99.18	99.94	96.53	99.32	98.71	98.41	92.64	91.61	70.79	94.20	92.10	86.99	76.23	77.27	70.71
	-TGLite	96.88	95.51	93.93	99.50	99.39	97.47	99.33	98.83	99.21	91.48	90.56	78.77	96.00	95.88	87.49	61.43	56.73	57.43
CAW	-Origin	98.19	98.54	94.30	98.35	98.59	93.93	74.24	77.24	62.18	96.26	96.63	90.53	Overtime			71.22	75.96	66.93
	-DyGLib	98.06	98.37	94.22	98.36	98.58	94.16	65.74	67.69	58.25	96.28	96.58	90.48	Overtime			72.19	75.80	67.63
DyREP	-Origin	91.71	92.14	82.93	96.67	96.54	90.67	69.97	65.05	60.07	92.14	91.09	70.94	OOM			OOM		
	-SPEED	90.87	91.48	82.76	96.47	96.50	88.88	76.17	72.44	70.22	88.41	86.85	74.01	96.30	95.82	91.03	OOM		
	-DyGLib	88.28	88.91	79.97	97.24	97.24	91.28	62.68	58.20	57.26	89.05	85.68	81.53	OOM			OOM		
DySAT	-Origin	86.68	89.87	65.13	91.79	93.45	71.95	45.51	43.36	48.62	OOM			OOM			OOM		
	-TGL	90.46	90.40	84.39	97.37	97.61	92.74	52.07	52.94	51.50	76.11	68.39	73.25	56.78	55.13	55.63	73.54	72.17	67.28

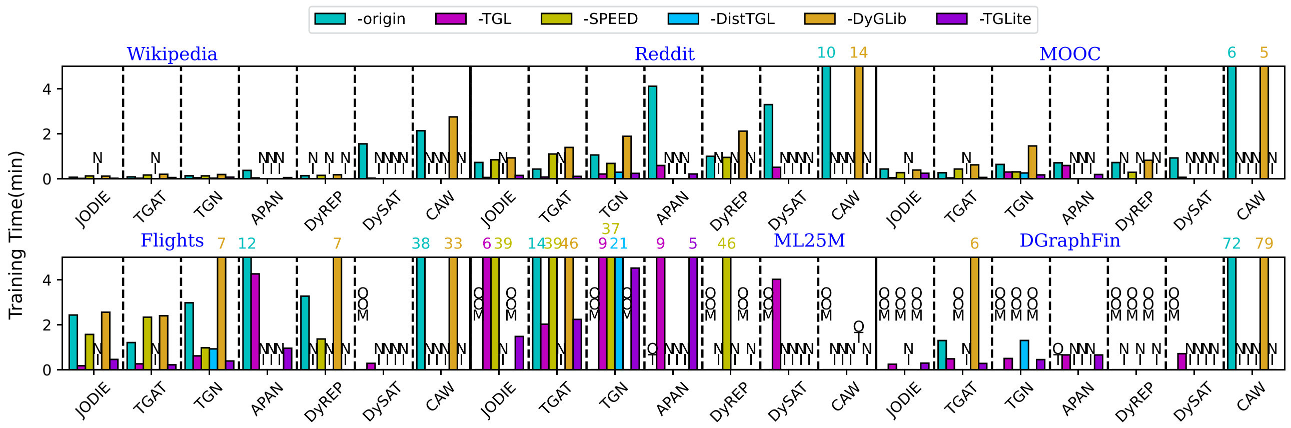

Fig. 10. Comparison of average per epoch training time of models and frameworks.

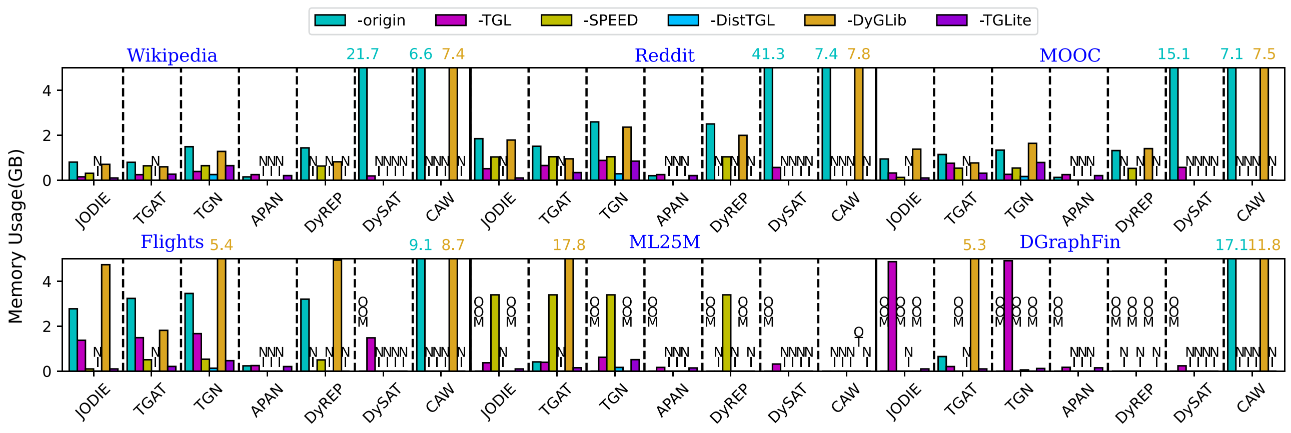

Fig. 11. Comparison of GPU memory usage of models and frameworks, with the GPU memory capacity equals 48 GB.

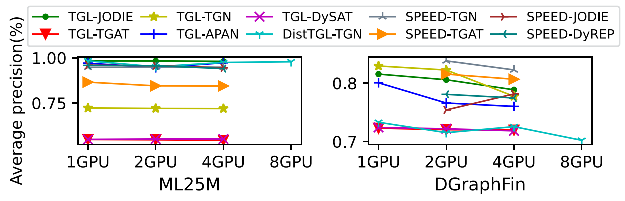

Fig. 12. Average precision for different numbers of GPUs.

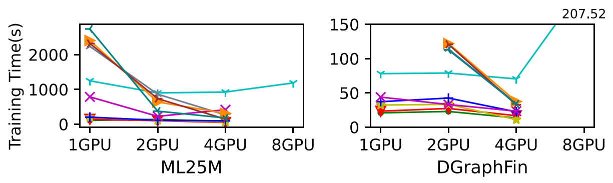

Fig. 13. Average training time per epoch for different numbers of GPUs.

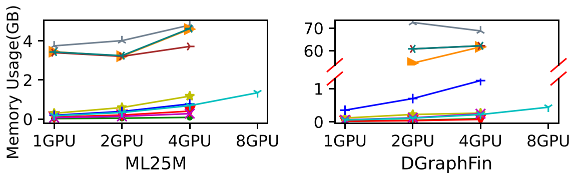

Fig. 14. GPU memory usage for different numbers of GPUs.

datasets, SPEED exhibits higher memory usage, possibly due to its graph partitioning approach. Frameworks often utilize specific techniques like TGL/DistTGL's T-CSR and SPEED's graph partitioning to accommodate large datasets. Despite this, the memory usage of TGL/DistTGL remains lower than that of SPEED, suggesting the efficiency of TGL/DistTGL's T-CSR. It's noteworthy that while ML25 M and DGraphFin have similar file sizes (as shown in Table III), training DGraphFin requires significantly more memory than ML25 M due to GNN sensitivity to node-related factors. TGLite integrates the T-CSR format of TGL with the deduplication optimization of TGOpt and therefore leads to a much smaller memory usage. Additionally, these frameworks do not maximize GPU memory usage, with most cases using only up to 4 GB.

D. *Multi-GPU Scalability*

Frameworks generally offer a key advantage over the origin models by incorporating a multi-GPU extension for faster parallel training. In this section, we analyze the performance of different frameworks with varying numbers of GPUs. We assess performance using 1 GPU, 2 GPUs on one machine, 4 GPUs on one machine, and 8 GPUs on two machines. Our analysis is based on the DGraphFin and ML25 M datasets, focusing on converged AP, average training time per epoch, and GPU memory usage. Results are presented in Figs. 12, 13, and 14, respectively. The lack of data for the 8GPU scenario is due to a lack of multi-machine training support, while the absence of 1GPU is due to out-of-memory issues. Note that among the five frameworks, only TGL, DistTGL, and SPEED are able to support multiple GPUs, and we report the corresponding results.

In ML25 M datasets, the AP values demonstrate stability with minimal variation as the number of GPUs increases, suggesting that multi-GPU parallel training has a limited impact on model

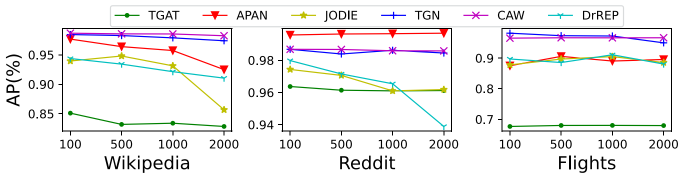

Fig. 15. Average precision for different batch sizes.

accuracy. An exception is seen with TGL, where a potential decrease in AP occurs as the number of GPUs rises, particularly noticeable in the DGraphFin dataset. This decline is attributed to TGL's reliance on mini-batch parallelism, which may overlook dependencies within one-time mini-batches, leading to decreased training accuracy when the total size of one-time mini-batches is doubled. While DistTGL generally shows resilience to changes in GPU numbers, there is a risk of performance deterioration when multiple machines are utilized, as evidenced in the DGraphFin results with 8 GPUs spread across two machines. the utilization of multiple machines will lead to accuracy challenges more easily. This suggests that DistTGL is relatively unaffected by GPU variations, potentially due to memory staleness and outdated information resulting from graph partitioning in distributed multi-machine environments. On the other hand, memory usage for models in the three frameworks generally increases with more GPUs, while the reduction in average per epoch training time is not as significant, except in cases like transitioning from 1 GPU to 2 GPUs in the ML25 M dataset. This trend is attributed to the impact of multi-GPU communication, which can contribute to an overall increase in training time. Interestingly, DistTGL experiences a rise in average per epoch training time when moving from 4 GPUs (within one machine) to 8 GPUs (across two machines), particularly with a threefold increase in the case of the DGraphFin dataset. This is primarily due to the high communication cost between multiple machines, resulting in longer training times for DistTGL under these configurations. Additionally, dynamic load imbalance from suboptimal graph partitioning strategies leads to increased synchronization waiting times, further extending overall training times. In future multi-machine endeavors, enhancing framework efficiency necessitates addressing key challenges such as implementing dynamic load balancing to reduce synchronization waiting times, optimizing remote data fetch to minimize communication overhead, and enhancing model structure for increased reliability in distributed environments.

E. *Hyperparameter Settings*

This section focuses on configuring different hyperparameters to assess their impacts on model training performance.

Impact of Batch Size: We experimented with different batch sizes (100, 500, 1000, and 2000) while maintaining a single GNN layer for the six CTDG models. Fig. 15 displays the models' AP results for the different batch sizes, revealing that a batch size of 100 consistently yields better performance. This suggests that, in general, smaller batch sizes can lead to improved indicator results by enabling more efficient feature learning. However, this efficiency may be counterbalanced by longer training times, as depicted in Fig. 16. Fig. 16 also indicates

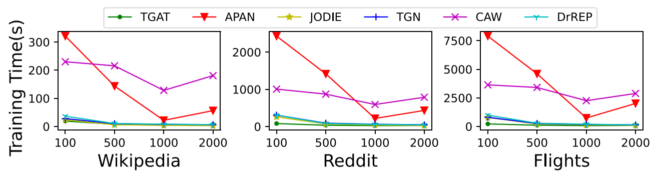

Fig. 16. Average training time per epoch for different batch sizes.

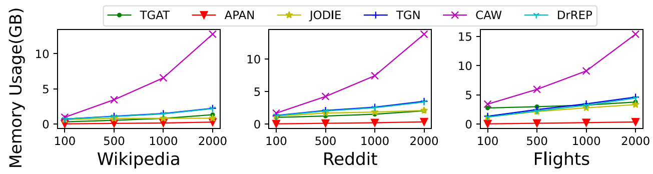

Fig. 17. GPU memory usage for different batch sizes.

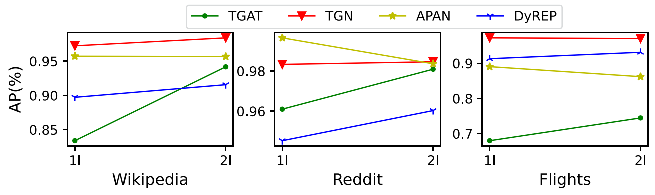

Fig. 18. Average precision for different GNN Layers.

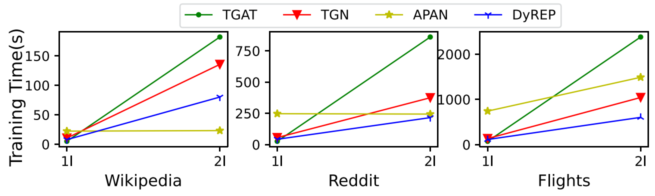

Fig. 19. Average training time per epoch for different GNN Layers.

that an optimal batch size lies between excessively large and excessively small values, representing a compromise. The choice of batch size is typically a trade-off between the number of batches (for smaller sizes) and the computational load per batch (for larger sizes), impacting operational speed. Although TGN, DyRep, and JODIE exhibit decreasing trends up to a batch size of 2000, it is anticipated that training time will escalate beyond a certain batch size threshold. Additionally, as shown in Fig. 17, there is a direct correlation between batch size and memory usage, with memory requirements increasing as the batch size grows.

Impact of GNN Layers: To investigate the impact of different numbers of GNN layers on model performance, we conducted a two-layer experiment with the TGAT, TGN, APAN, and DyREP models. JODIE was excluded due to the absence of layer concepts, and CAW was prone to OOM issues with two layers. The experiments were conducted with a batch size of 1000. The results in Fig. 18. The findings suggest that increasing the number of GNN layers can enhance performance. However, as illustrated in Figs. 19 and 20, this improvement comes at the cost of increased training time and memory consumption. The objective is to increase the number of GNN layers to improve performance within a reasonable epoch time while mitigating excessive time and OOM concerns. Notably, APAN appears less sensitive to the number of layers, as its model architecture is

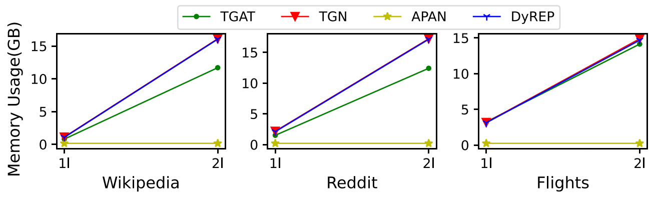

Fig. 20. GPU memory usage for different GNN Layers.

TABLE VIII
THE NODE CLASSIFICATION TASK ON CTDG AND DTDG. WE USE AUC(%) TO EVALUATE THE TASK

Type	Model	Wikipedia	Reddit	MOOC
		AUC	AUC	AUC
CTDG	JODIE	80.21	63.92	60.99
	TGN	86.85	68.21	53.86
	DyREP	83.57	53.10	65.61
	TGAT	85.71	67.24	6311
	APAN	87.01	55.90	64.12
CTDG	EvolveGCN-H	69.81	59.27	67.92
	EvolveGCN-O	79.20	59.64	67.82

not heavily dependent on this factor. Consequently, augmenting the number of layers does not substantially inflate training time or memory usage for APAN. While the results for the three layers are not presented, TGAT, TGN, and DyREP encountered OOM issues with three layers, whereas APAN did not. This discrepancy can be attributed to APAN's model architecture exhibiting minimal correlation with the number of GNN layers.

F. Node Classification Evaluation

In this section, we assess the performance of various dynamic GNN models for the node classification task. Specifically, we consider the JODIE, TGN, DyREP, TGAT, and APAN of CTDG models, as well as the EvolveGCN of the DTDG model. These models are evaluated on the Wikipedia, Reddit, and MOOC datasets, using appropriate labels for node classification. A single GNN layer with a batch size of 1000 is utilized for consistency across all models. The results of the evaluation are presented in Table VIII. Interestingly, the AUC metric values for these dynamic GNN models are notably lower in node classification tasks, particularly for the Reddit and MOOC datasets. This disparity in performance may be attributed to the fact that these models were primarily designed for the more prevalent link prediction task in dynamic graphs. Surprisingly, there does not appear to be a significant discrepancy between the performance of DTDG and CTDG models in node classification tasks, suggesting that both kinds of dynamic GNN models are equally effective in this context.

VII. OPEN CHALLENGES

Compared to static graphs, dynamic graphs introduce temporal information, necessitating additional time modules in models, which increases complexity and poses challenges for downstream tasks. For example, dynamic link prediction requires precise timing considerations, making it more difficult compared to static graphs. However, effectively capturing real-world data evolution over time requires overcoming this challenge. Despite progress in dynamic GNN models and training frameworks, open challenges persist, with ongoing exploration needed to address limitations in diversity, accuracy, efficiency, and scalability.

Diversity in Application Domains. Dynamic graph learning is applied in diverse domains such as social networks, transportation networks, epidemic transmission, and recommendation systems, as discussed in Section II-A. However, each specific application has unique dynamic graph characteristics, highlighting the need for specialized methods to effectively address the diverse requirements of individual scenarios.

Requirement for a Unified Framework. Existing dynamic graph algorithms utilize various methods to capture time dependencies within graph structures, making it challenging to establish a unified framework. While efforts like TGL [138] have strived to create a unified graph operator, they typically cover only a restricted range of models. Furthermore, the implementation of the model is determined by the configuration file. In contrast, TGLite provides functions that can be used, and is the direction expected. Developing a comprehensive unified graph operator capable of encompassing a majority of algorithms is a crucial yet formidable endeavor in the field of dynamic graph learning.

Challenges in Processing Dynamic Graph Updates. While existing dynamic GNN frameworks provide functional support for training modules essential for dynamic graph models, their assistance is constrained, especially in dynamic graph data storage. Current frameworks commonly utilize structures like CSR or T-CSR, which encounter difficulties in promptly supporting graph updates to real-time dynamic graph updates. This constraint also restricts the adaptability of dynamic graph models. A more practical and efficient dynamic graph storage format is needed.

Challenges in Storing Large-Scale Dynamic Graphs. The data volume of dynamic graph data evolving over time is considerably larger than that of static graphs. Dynamic GNN models also necessitate the storage of additional long-term and short-term memory information related to the evolution of graph data, resulting in significant storage and computational requirements. Many existing models are limited in their ability to handle large graphs, and when confronted with such scenarios, they often require substantial GPU resources for processing, indicating scalability challenges. As the size of graphs increases, distributed processing becomes a viable solution [139]. Therefore, there is an urgent need to advance distributed and parallel computing methodologies to effectively manage and process large-scale dynamic graph data.

Inefficient Training Data Extraction. In dynamic GNN training, the extraction of training data involves not only feature data but also historical memory information, which can impede GNN training efficiency. In distributed parallel training frameworks, data extraction also includes data transmission among multiple GPUs and machines, adding to the inefficiency of the process. Furthermore, updating this memory information post-training further reduces the effectiveness of data extraction in dynamic GNN training. On the other hand, our experiments reveal instances of under-utilization of GPU memory, presenting an opportunity to leverage this available memory space to accelerate the data extraction process.

Challenges in Distributed Parallel Training Efficiency. Current frameworks are limited to training on a single machine with single or multiple GPUs, lacking support for distributed environments across multiple machines. This restricts scalability in parallel training large dynamic graphs. The complexity of dynamic GNN models and temporal dependencies pose obstacles to efficient parallel training. Existing frameworks may overlook dependencies within mini-batches or focus solely on temporal dependencies, leading to suboptimal efficiency.

Challenges in Supporting Accelerator-Heterogeneity. Integrating different types of accelerators in distributed systems poses hurdles for optimizing dynamic GNN training. Varying computational capabilities and memory constraints lead to workload imbalances. Mixed interconnects also cause communication bottlenecks, hindering data synchronization. Accelerators operating at different speeds create inefficiencies. Adaptive workload schedulers, hybrid communication protocols, and hardware-agnostic abstractions are needed to unify execution across accelerators while maintaining temporal integrity in DGNN training.

VIII. Conclusion

This paper provides a comprehensive comparative analysis and experimental evaluation of dynamic GNNs. It covers 91 dynamic GNN models with a novel taxonomy, compares 17 dynamic GNN training frameworks, and includes commonly used benchmarks for evaluating dynamic GNNs. The paper also includes extensive experiments on ten representative models and five frameworks across six standard graph datasets using unified benchmarks and evaluation metrics. Performance evaluation covers metrics related to convergence accuracy, training efficiency, and GPU memory usage, considering both single-GPU and multiple-GPU scenarios. The analysis and evaluation results highlight various open challenges in the dynamic GNN field to offer valuable principles for future researchers to enhance the generality, performance, efficiency, and scalability of dynamic GNN models and frameworks.

Acknowledgment

The author gratefully acknowledges the support of Zhejiang University Education Foundation Qizhen Scholar Foundation.

References

[1] S. Shreif, B. Angela, V. Hannes, and I. Alexandru, “The future is big graphs: A community view on graph processing systems,” *Commun. ACM*, vol. 64, no. 9, pp. 62–71, 2021.

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

[3] W. Fan et al., “A graph neural network framework for social recommendations,” *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 5, pp. 2033–2047, May 2022.

[4] H. Yuan et al., “Comprehensive evaluation of GNN training systems: A data management perspective,” *Proc. VLDB Endowment*, vol. 17, no. 6, pp. 1241–1254, 2024.

[5] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[6] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.

[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–12.

[8] M. Zhang and Y. Chen, “Link prediction based on graph neural networks,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5171–5181.

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

[10] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, “A gentle introduction to graph neural networks,” *Distill*, vol. 6, 2021, Art. no. e33.

[11] A. Paranjape, A. R. Benson, and J. Leskovec, “Motifs in temporal networks,” in *Proc. ACM Int. Conf. Web Search Data Mining*, 2017, pp. 601–610.

[12] A. Pareja et al., “EvolveGCN: Evolving graph convolutional networks for dynamic graphs,” in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 5363–5370.

[13] L. Zhao et al., “T-GCN: A temporal graph convolutional network for traffic prediction,” *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3848–3858, Sep. 2020.

[14] S. Kumar, X. Zhang, and J. Leskovec, “Predicting dynamic embedding trajectory in temporal interaction networks,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1269–1278.

[15] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, “Temporal graph networks for deep learning on dynamic graphs,” in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1–16.

[16] S. Deng, H. Rangwala, and Y. Ning, “Learning dynamic context graphs for predicting social events,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1007–1016.

[17] S. M. Kazemi et al., “Representation learning for dynamic graphs: A survey,” *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 2648–2720, 2020.

[18] B. Yu, H. Yin, and Z. Zhu, “Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting,” in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 3634–3640.

[19] B. He, X. He, Y. Zhang, R. Tang, and C. Ma, “Dynamically expandable graph convolution for streaming recommendation,” in *Proc. ACM Web Conf.*, 2023, pp. 1457–1467.

[20] C. Zhang et al., “A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data,” in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 1409–1416.

[21] Z. Zheng, J. Shao, J. Zhu, and H. T. Shen, “Relational temporal graph convolutional networks for ranking-based stock prediction,” in *Proc. IEEE Int. Conf. Data Eng.*, 2023, pp. 123–136.

[22] H. Zhang et al., “TFE-GNN: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification,” in *Proc. ACM Web Conf.*, 2023, pp. 2066–2075.

[23] Y. Wang, P. Li, C. Bai, V. Subrahmanian, and J. Leskovec, “Generic representation learning for dynamic social interaction,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 1–9.

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

[25] J. Wu, M. Cao, J. C. K. Cheung, and W. L. Hamilton, “TeMP: Temporal message passing for temporal knowledge graph completion,” in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2020, pp. 5730–5746.

[26] P. Zhang et al., “Continual learning on dynamic graphs via parameter isolation,” in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2023, pp. 601–611.

[27] S. Bonner et al., “Temporal neighbourhood aggregation: Predicting future links in temporal graphs via recurrent variational graph convolutions,” in *Proc. IEEE Int. Conf. Big Data*, 2019, pp. 5336–5345.

[28] M. Yang, M. Zhou, M. Kalander, Z. Huang, and I. King, “Discrete-time temporal network embedding via implicit hierarchical learning in hyperbolic space,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2021, pp. 1975–1985.

[29] X. Qin, N. Sheikh, C. Lei, B. Reinwald, and G. Domeniconi, “SEIGN: A simple and efficient graph neural network for large dynamic graphs,” in *Proc. IEEE Int. Conf. Data Eng.*, 2023, pp. 2850–2863.

[30] A. Cini, I. Marisca, F. M. Bianchi, and C. Alippi, “Scalable spatiotemporal graph neural networks,” in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 7218–7226.

[31] J. Li et al., “Scaling up dynamic graph representation learning via spiking neural networks,” in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 8588–8596.

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

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

[34] Y. Zhu, F. Lyu, C. Hu, X. Chen, and X. Liu, “Encoder-decoder architecture for supervised dynamic graph learning: A survey,” 2022, *arXiv:2203.10480*.

[35] G. Jin et al., “Spatio-temporal graph neural networks for predictive learning in urban computing: A survey,” *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 10, pp. 5388–5408, Oct. 2024.

[36] A. Longa et al., “Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities,” 2023, *arXiv:2302.01018*.

[37] X. Huang et al., “DGraph: A large-scale financial dataset for graph anomaly detection,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 22765–22777.

[38] X. Yang, Y. Sun, X. Chen, Y. Zhang, and X. Yuan, “Graph structure learning for spatial-temporal imputation: Adapting to node and feature scales,” in *Proc. AAAI Conf. Artif. Intell.*, 2025, pp. 959–967.

[39] L. Cao, B. Wang, G. Jiang, Y. Yu, and J. Dong, “Spatiotemporal-aware trend-seasonality decomposition network for traffic flow forecasting,” in *Proc. AAAI Conf. Artif. Intell.*, 2025, pp. 11463–11471.

[40] R. Trivedi, H. Dai, Y. Wang, and L. Song, “Know-evolve: Deep temporal reasoning for dynamic knowledge graphs,” in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3462–3471.

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

[42] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, “Inductive representation learning on temporal graphs,” in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–19.

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

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

[45] H. Jaeger, “Echo state network,” *Scholarpedia*, vol. 2, no. 9, 2007, Art. no. 2330.

[46] S.-H. Yang and H. Zha, “Mixture of mutually exciting processes for viral diffusion,” in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1–9.

[47] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” 2018, *arXiv:1803.01271*.

[48] A. Rahimi and B. Recht, “Random features for large-scale kernel machines,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2007, pp. 1177–1184.

[49] W. Hu, Y. Yang, Z. Cheng, C. Yang, and X. Ren, “Time-series event prediction with evolutionary state graph,” in *Proc. ACM Int. Conf. Web Search Data Mining*, 2021, pp. 580–588.

[50] K. Liu, F. Zhao, G. Xu, X. Wang, and H. Jin, “RETIA: Relation-entity twin-interact aggregation for temporal knowledge graph extrapolation,” in *Proc. IEEE Int. Conf. Data Eng.*, 2023, pp. 1761–1774.

[51] K. Liang et al., “Learn from relational correlations and periodic events for temporal knowledge graph reasoning,” in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2023, pp. 1559–1568.

[52] A. Vaswani et al., “Attention is all you need,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[53] J. Li et al., “Predicting path failure in time-evolving graphs,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1279–1289.

[54] M. Schlichtkrull et al., “Modeling relational data with graph convolutional networks,” in *Proc. Eur. Semantic Web Conf.*, 2018, pp. 593–607.

[55] R. Jiang et al., “Spatio-temporal meta-graph learning for traffic forecasting,” in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 8078–8086.

[56] G. Casadesus-Vila, J.-A. Ruiz-de Azua, and E. Alarcon, “Toward autonomous cooperation in heterogeneous nanosatellite constellations using dynamic graph neural networks,” 2024, *arXiv:2403.00692*.

[57] J. Liu, X. Shang, X. Han, W. Zhang, and H. Yin, “Spatial-temporal memories enhanced graph autoencoder for anomaly detection in dynamic graphs,” 2024, *arXiv:2403.09039*.

[58] G. Jin, L. Liu, F. Li, and J. Huang, “Spatio-temporal graph neural point process for traffic congestion event prediction,” in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 14268–14276.

[59] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, “Structured sequence modeling with graph convolutional recurrent networks,” in *Proc. Int. Conf. Neural Inf. Process.*, 2018, pp. 362–373.

[60] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[61] E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou, and X. Qian, "Variational graph recurrent neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 10701–10711.

[62] J. Chen et al., "E-LSTM-D: A deep learning framework for dynamic network link prediction," *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 51, no. 6, pp. 3699–3712, Jun. 2021.

[63] H. Li, Z. Zhang, D. Liang, and Y. Jiang, "K-truss based temporal graph convolutional network for dynamic graphs," in *Proc. Asian Conf. Mach. Learn.*, PMLR, 2024, pp. 739–754.

[64] D. Chen, S. Zheng, M. Xu, Z. Zhu, and Y. Zhao, "SiGNN: A spike-induced graph neural network for dynamic graph representation learning," *Pattern Recognit.*, vol. 158, 2025, Art. no. 111026.

[65] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[66] H. Liu et al., "TodyNet: Temporal dynamic graph neural network for multivariate time series classification," *Inf. Sci.*, vol. 677, 2024, Art. no. 120914.

[67] G. Bai, C. Ling, and L. Zhao, "Temporal domain generalization with drift-aware dynamic neural networks," in *Proc. Int. Conf. Learn. Representations*, 2023, pp. 1–19.

[68] F. Zhou, X. Xu, C. Li, G. Trajcevski, T. Zhong, and K. Zhang, "A heterogeneous dynamical graph neural networks approach to quantify scientific impact," 2020, *arXiv:2003.12042*.

[69] H. Xue, L. Yang, W. Jiang, Y. Wei, Y. Hu, and Y. Lin, "Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal RNN," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2021, pp. 282–298.

[70] H. Qian et al., "MDGNN: Multi-relational dynamic graph neural network for comprehensive and dynamic stock investment prediction," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 14642–14650.

[71] J. Wang, W. Zhu, G. Song, and L. Wang, "Streaming graph neural networks with generative replay," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2022, pp. 1878–1888.

[72] M. Salvaris, D. Dean, and W. H. Tok, "Generative adversarial networks," in *Deep Learning with Azure: Building and Deploying Artificial Intelligence Solutions on the Microsoft AI Platform*. Springer, 2018, pp. 187–208.

[73] J. Hu, Y. Liang, Z. Fan, H. Chen, Y. Zheng, and R. Zimmermann, "Graph neural processes for spatio-temporal extrapolation," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2023, pp. 752–763.

[74] M. Garnelo et al., "Neural processes," 2018, *arXiv:1807.01622*.

[75] Z. Zhang et al., "Spectral invariant learning for dynamic graphs under distribution shifts," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2024, pp. 6619–6633.

[76] J. You, T. Du, and J. Leskovec, "ROLAND: Graph learning framework for dynamic graphs," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2022, pp. 2358–2366.

[77] Y. Zhu et al., "WinGNN: Dynamic graph neural networks with random gradient aggregation window," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2023, pp. 3650–3662.

[78] Q. Yang, C. Ma, Q. Zhang, X. Gao, C. Zhang, and X. Zhang, "Interpretable research interest shift detection with temporal heterogeneous graphs," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2023, pp. 321–329.

[79] Y. Fang et al., "When spatio-temporal meet wavelets: Disentangled traffic forecasting via efficient spectral graph attention networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2023, pp. 517–529.

[80] M. Yang et al., "Vehicle interactive dynamic graph neural network based trajectory prediction for internet of vehicles," *IEEE Int. Things J.*, vol. 11, no. 22, pp. 35777–35790, Nov. 2024.

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

[82] P. Goyal, S. R. Chhetri, and A. Canedo, "Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowl.-Based Syst.*, vol. 187, 2020, Art. no. 104816.

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

[84] L. Qian, Q. Zuo, H. Liu, and H. Zhu, "Multivariate time series classification based on spatial-temporal attention dynamic graph neural network," *Appl. Intell.*, vol. 55, no. 2, pp. 1–18, 2025.

[85] J. Layne, J. Carpenter, E. Serra, and F. Gullo, "Temporal SIR-GN: Efficient and effective structural representation learning for temporal graphs," *Proc. VLDB Endowment*, vol. 16, no. 9, pp. 2075–2089, 2023.

[86] M. Joaristi and E. Serra, "SIR-GN: A fast structural iterative representation learning approach for graph nodes," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 6, pp. 1–39, 2021.

[87] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 571–578.

[88] H. Huang, J. Tang, L. Liu, J. Luo, and X. Fu, "Triadic closure pattern analysis and prediction in social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 12, pp. 3374–3389, Dec. 2015.

[89] A. Bastos, A. Nadgeri, K. Singh, T. Suzumura, and M. Singh, "Learnable spectral wavelets on dynamic graphs to capture global interactions," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 6779–6787.

[90] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "The spectral graph wavelet transform: Fundamental theory and fast computation," in *Vertex-Frequency Analysis of Graph Signals*, Berlin, Germany: Springer, 2018, pp. 141–175.

[91] S. Zheng, H. Yin, T. Chen, Q. V. H. Nguyen, W. Chen, and L. Zhao, "DREAM: Adaptive reinforcement learning based on attention mechanism for temporal knowledge graph reasoning," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2023, pp. 1578–1588.

[92] H. Yuan et al., "Environment-aware dynamic graph learning for out-of-distribution generalization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2024, pp. 49715–49747.

[93] Y. Wang, P. Li, C. Bai, and J. Leskovec, "TEDIC: Neural modeling of behavioral patterns in dynamic social interaction networks," in *Proc. ACM Web Conf.*, 2021, pp. 693–705.

[94] S. Khodabandehlou and A. H. Golpayegani, "FiFrauD: Unsupervised financial fraud detection in dynamic graph streams," *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 5, pp. 1–29, 2024.

[95] K. Zhao and L. Zhang, "Causality-inspired spatial-temporal explanations for dynamic graph neural networks," in *Proc. Int. Conf. Learn. Representations*, 2024, pp. 1–13.

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

[97] H. Tang, S. Wu, G. Xu, and Q. Li, "Dynamic graph evolution learning for recommendation," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2023, pp. 1589–1598.

[98] L. Fang, K. Feng, J. Gui, S. Feng, and A. Hu, "Anonymous edge representation for inductive anomaly detection in dynamic bipartite graph," *Proc. VLDB Endowment*, vol. 16, no. 5, pp. 1154–1167, 2023.

[99] S. Fan, S. Zhang, X. Wang, and C. Shi, "Directed acyclic graph structure learning from dynamic graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 7512–7521.

[100] Y. Wu, Y. Fang, and L. Liao, "On the feasibility of simple transformer for dynamic graph modeling," in *Proc. ACM Web Conf.*, 2024, pp. 870–880.

[101] L. Yu, L. Sun, B. Du, and W. Lv, "Towards better dynamic graph learning: New architecture and unified library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2024, pp. 67686–67700.

[102] Z. Zhang, Z. Zhang, X. Wang, Y. Qin, Z. Qin, and W. Zhu, "Dynamic heterogeneous graph attention neural architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 11307–11315.

[103] H. Li, C. Li, K. Feng, Y. Yuan, G. Wang, and H. Zha, "Robust knowledge adaptation for dynamic graph neural networks," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 11, pp. 6920–6933, Nov. 2024.

[104] W. Jin, M. Qu, X. Jin, and X. Ren, "Recurrent event network: Autoregressive structure inference over temporal knowledge graphs," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2020, pp. 6669–6683.

[105] J. You, Y. Wang, A. Pal, P. Eksombatchai, C. Rosenburg, and J. Leskovec, "Hierarchical temporal convolutional networks for dynamic recommender systems," in *Proc. ACM Web Conf.*, 2019, pp. 2236–2246.

[106] D. Tortorella et al., "Dynamic graph echo state networks," in *Proc. 29th Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn. (ESANN 2021)*, 2021, pp. 99–104, Paper i6doc.

[107] C.-Y. Sang, J.-J. Chen, and S.-G. Liao, "DyHGTCR-Cas: Learning unified spatio-temporal features based on dynamic heterogeneous graph neural network for information cascade prediction," *Inf. Process. Manage.*, vol. 62, no. 3, 2025, Art. no. 104029.

[108] X. Li, M. Zhang, S. Wu, Z. Liu, L. Wang, and S. Y. Philip, "Dynamic graph collaborative filtering," in *Proc. IEEE Int. Conf. Data Mining*, 2020, pp. 322–331.

[109] K. Feng, C. Li, X. Zhang, and J. Zhou, "Towards open temporal graph neural networks," in *Proc. Int. Conf. Learn. Representations*, 2023, pp. 1–23.
[110] S. Suresh, M. Shrivastava, A. Mukherjee, J. Neville, and P. Li, "Expressive and efficient representation learning for ranking links in temporal graphs," in *Proc. ACM Web Conf.*, 2023, pp. 567–577.
[111] Y. Tian, Y. Qi, and F. Guo, "FreeDyG: Frequency enhanced continuous time dynamic graph model for link prediction," in *Proc. Int. Conf. Learn. Representations*, 2024, pp. 1–20.
[112] D. Chen et al., "Guiding fusion of dynamic functional and effective connectivity in spatio-temporal graph neural network for brain disorder classification," *Knowl.-Based Syst.*, vol. 309, 2025, Art. no. 112856.
[113] Y. Luo and P. Li, "Neighborhood-aware scalable temporal network representation learning," in *Proc. Learn. Graphs Conf.*, 2022, pp. 1–18.
[114] Y. Zhang et al., "TIGER: Temporal interaction graph embedding with restarts," in *Proc. ACM Web Conf.*, 2023, pp. 478–488.
[115] J. Su, D. Zou, and C. Wu, "PRES: Toward scalable memory-based dynamic graph neural networks," 2024, *arXiv:2402.04284*.
[116] S. Gao, Y. Li, Y. Shen, Y. Shao, and L. Chen, "ETC: Efficient training of temporal graph neural networks over large-scale dynamic graphs," *Proc. VLDB Endowment*, vol. 17, no. 5, pp. 1060–1072, 2024.
[117] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "DyRep: Learning representations over dynamic graphs," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–25.
[118] Y. Lu, X. Wang, C. Shi, P. S. Yu, and Y. Ye, "Temporal network embedding with micro- and macro-dynamics," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2019, pp. 469–478.
[119] Z. Han, J. Jiang, Y. Wang, Y. Ma, and V. Tresp, "The graph hawkes network for reasoning on temporal knowledge graphs," 2020, *arXiv:2003.13432*.
[120] B. Knyazev, C. Augusta, and G. W. Taylor, "Learning temporal attention in dynamic graphs with bilinear interactions," *Public Library Sci.*, vol. 16, no. 3, 2021, Art. no. e0247936.
[121] Z. Zhao et al., "Time-interval aware share recommendation via bi-directional continuous time dynamic graphs," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2023, pp. 822–831.
[122] Y. Wang, Y.-Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–22.
[123] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Dynamic network embeddings: From random walks to temporal random walks," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 1085–1092.
[124] X. Wang et al., "APAN: Asynchronous propagation attention network for real-time temporal graph embedding," in *Proc. ACM Int. Conf. Manage. Data*, 2021, pp. 2628–2638.
[125] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 719–728.
[126] C. Wu et al., "Instant representation learning for recommendation over large dynamic graphs," in *Proc. IEEE Int. Conf. Data Eng.*, 2023, pp. 82–95.
[127] L. Han, R. Zhang, L. Sun, B. Du, Y. Fu, and T. Zhu, "Generic and dynamic graph representation learning for crowd flow modeling," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 4293–4301.
[128] J. Liu, J. Liu, K. Zhao, Y. Tang, and W. Chen, "TP-GNN: Continuous dynamic graph neural network for graph classification," in *Proc. IEEE Int. Conf. Data Eng.*, 2024, pp. 2848–2861.
[129] S. Ji et al., "Community-based dynamic graph learning for popularity prediction," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2023, pp. 930–940.
[130] S. Ji, M. Liu, L. Sun, C. Liu, and T. Zhu, "MemMap: An adaptive and latent memory structure for dynamic graph learning," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2024, pp. 1257–1268.
[131] B. Rozemberczki et al., "PyTorch geometric temporal: Spatiotemporal signal processing with neural machine learning models," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2021, pp. 4564–4573.
[132] V. T. Chakaravarthy, S. S. Pandian, S. Raje, Y. Sabharwal, T. Suzumura, and S. Ubaru, "Efficient scaling of dynamic graph neural networks," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, pp. 1–15.
[133] C. Wang, D. Sun, and Y. Bai, "PiPAD: Pipelined and parallel dynamic GNN training on GPUs," in *Proc. ACM SIGPLAN Annu. Symp. Princ. Pract. Parallel Program.*, 2023, pp. 405–418.
[134] M. Guan, A. P. Iyer, and T. Kim, "DynaGraph: Dynamic graph neural networks at scale," in *Proc. ACM SIGMOD Joint Int. Workshop Graph Data Manage. Exp. Syst., Netw. Data Analytics*, 2022, pp. 1–10.
[135] K. Fu, Q. Chen, Y. Yang, J. Shi, C. Li, and M. Guo, "BLAD: Adaptive load balanced scheduling and operator overlap pipeline for accelerating the dynamic GNN training," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2023, pp. 1–13.
[136] F. Chen, P. Li, and C. Wu, "DGC: Training dynamic graphs with spatio-temporal non-uniformity using graph partitioning by chunks," *Proc. ACM Manage. Data*, vol. 1, no. 4, pp. 1–25, 2023.
[137] J. M. Cherian, N. P. Manoj, K. J. Concessao, and U. Cheramangalath, "STGraph: A framework for temporal graph neural networks," in *Proc. Int. Parallel Distrib. Process. Symp. Workshops*, 2024, pp. 496–505.
[138] H. Zhou, D. Zheng, I. Nisa, V. Ioannidis, X. Song, and G. Karypis, "TGL: A general framework for temporal GNN training on billion-scale graphs," *Proc. VLDB Endowment*, vol. 15, no. 8, pp. 1572–1580, 2022.
[139] H. Zhou, D. Zheng, X. Song, G. Karypis, and V. Prasanna, "DistTGL: Distributed memory-based temporal graph neural network training," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2023, pp. 1–12.
[140] Z. Fang, Q. Sun, Q. Wang, L. Chen, and Y. Gao, "Distributed temporal graph neural network learning over large-scale dynamic graphs," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2024, pp. 51–66.
[141] Y. Wang and C. Mendis, "TGLite: A lightweight programming framework for continuous-time temporal graph neural networks," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2024, pp. 1183–1199.
[142] X. Chen et al., "SPEED: Streaming partition and parallel acceleration for temporal interaction graph embedding," 2023, *arXiv:2308.14129*.
[143] C. Chen et al., "NeutronStream: A dynamic GNN training framework with sliding window for graph streams," *Proc. VLDB Endowment*, vol. 17, no. 3, pp. 455–468, 2023.
[144] Y. Zhong, G. Sheng, T. Qin, M. Wang, Q. Gan, and C. Wu, "GNNFlow: A distributed framework for continuous temporal GNN learning on dynamic graphs," 2023, *arXiv:2311.17410*.
[145] G. Sheng, J. Su, C. Huang, and C. Wu, "MSPipe: Efficient temporal GNN training via staleness-aware pipeline," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2024, pp. 2651–2662.
[146] Y. Xia et al., "Redundancy-free and load-balanced TGNN training with hierarchical pipeline parallelism," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 11, pp. 1904–1919, Nov. 2024.
[147] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch geometric," 2019, *arXiv:1903.02428*.
[148] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–18.
[149] L. Ma et al., "NeuGraph: Parallel deep neural network computation on large graphs," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2019, pp. 443–458.
[150] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of graph neural networks with ROC," *Mach. Learn. Syst.*, vol. 2, pp. 187–198, 2020.
[151] R. Zhu et al., "AliGraph: A comprehensive graph neural network platform," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 2094–2105, 2019.
[152] X. Zhang, Y. Shen, Y. Shao, and L. Chen, "DUCATI: A dual-cache training system for graph neural networks on giant graphs with the GPU," *Proc. ACM Manage. Data*, vol. 1, no. 2, pp. 1–24, 2023.
[153] S. Gandhi and A. P. Iyer, "P3: Distributed deep graph learning at scale," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2021, pp. 551–568.
[154] C. Wan, Y. Li, C. R. Wolfe, A. Kyrillidis, N. S. Kim, and Y. Lin, "PipeGCN: Efficient full-graph training of graph convolutional networks with pipelined feature communication," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–24.
[155] X. Ai et al., "NeutronOrch: Rethinking sample-based GNN training under CPU-GPU heterogeneous environments," 2023, *arXiv:2311.13225*.
[156] H. Zeng and V. Prasanna, "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2020, pp. 255–265.
[157] J. Hu, Y. Zou, X. Song, and Y. Liu, "GALA-GNN: Optimization for training GNNs of large-scale graphs on heterogeneous platforms," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl.*, 2024, pp. 460–467.

[158] Y. Wang and C. Mendis, "TGOpt: Redundancy-aware optimizations for temporal graph attention networks," in *Proc. ACM SIGPLAN Annu. Symp. Princ. Pract. Parallel Program.*, 2023, pp. 354–368.
[159] A. Pavlisic, "The wikipedia/reddit/mooc/lastfm datasets," 2009. Retrieved Apr. 28, 2024. [Online]. Available: http://snap.stanford.edu/jodie
[160] Anon, "The dgraphfin datasets," 2022. Retrieved Apr. 28, 2024. [Online]. Available: https://dgraph.xinye.com/dataset
[161] F. Poursafaei, S. Huang, K. Pelrine, and R. Rabbany, "The enron/flights datasets," 2022. Retrieved Apr. 28, 2024. [Online]. Available: https://zenodo.org/records/7213796#.Y1cO6y8r30o
[162] R. A. Rossi and N. K. Ahmed, "The Facebook datasets," 2015. Retrieved Apr. 28, 2024. [Online]. Available: https://networkrepository.com/fb-wosn-friends.php
[163] A. Madan et al., "The social evolution datasets," 2008. Retrieved Apr. 28, 2024. [Online]. Available: http://realitycommons.media.mit.edu/socialevolution.html
[164] J. Kunegis, "The UCI datasets," 2017. Retrieved Apr. 28, 2024. [Online]. Available: http://konect.cc/networks/opsahl-ucforum/
[165] Anon, "The ml25m datasets," 2019. Retrieved Apr. 28, 2024. [Online]. Available: https://grouplens.org/datasets/movielens/25m/
[166] R. A. Rossi and N. K. Ahmed, "The fb-forum datasets," 2015. Retrieved Apr. 28, 2024. [Online]. Available: https://networkrepository.com/fb-forum.php
[167] Anon, "The DBLP datasets," 2022. Retrieved Apr. 28, 2024. [Online]. Available: https://www.dropbox.com/sh/palzyh5box1uc1v/AACSLHB7PChT-ruN-rksZTCYa?dl=0
[168] Anon, "The yelp datasets," 2004. Retrieved Apr. 28, 2024. [Online]. Available: https://www.yelp.com/dataset
[169] Y. Liu, H. Li, A. Garcia-Duran, M. Niepert, D. Onoro-Rubio, and D. S. Rosenblum, "The ICEWS14/05–15 datasets," 2018. Retrieved Apr. 28, 2024. [Online]. Available: https://github.com/mniepert/mmkb/tree/master/TemporalKGs
[170] E. Boschee, J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward, "ICEWS coded event data," 2015. [Online]. Available: https://doi.org/10.7910/DVN/28075
[171] Anon, "The GDELT datasets," 2022. Retrieved Apr. 28, 2024. [Online]. Available: https://github.com/amazon-science/tgl/blob/main/down.sh
[172] A. Pavlisic, "The bitcoin-alpah/bitcoin-otc/as-773 datasets," 2009. Retrieved Apr. 28, 2024. [Online]. Available: https://snap.stanford.edu/data
[173] B. Klimt and Y. Yang, "Introducing the enron corpus," in *Proc. CEAS*, 2004, pp. 92–96.
[174] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *Proc. ACM Workshop Online Social Netw.*, 2009, pp. 37–42.
[175] A. Madan et al., "Sensing the "health state" of a community," *IEEE Pervasive Comput.*, vol. 11, no. 4, pp. 36–45, Fourth Quarter 2012.
[176] P. Panzarasa, T. Opsahl, and K. M. Carley, "Patterns and dynamics of users' behavior and interaction: Network analysis of an online community," *J. Amer. Soc. Inf. Sci. Technol.*, vol. 60, no. 5, pp. 911–932, 2009.
[177] F. M. Harper and J. A. Konstan, "The movieLens datasets: History and context," *ACM Trans. Interactive Intell. Syst.*, vol. 5, no. 4, pp. 1–19, 2015.
[178] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 4292–4293.
[179] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "ArnetMiner: Extraction and mining of academic social networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2008, pp. 990–998.
[180] Z. Han, P. Chen, Y. Ma, and V. Tresp, "Explainable subgraph reasoning for forecasting on temporal knowledge graphs," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–24.
[181] K. Leetaru and P. A. Schrodt, "GDELT: Global data on events, location, and tone, 1979–2012," *ISA Annu. Conv.*, vol. 2, no. 4, pp. 1–49, 2013.
[182] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. Subrahmanian, "REV2: Fraudulent user prediction in rating platforms," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2018, pp. 333–341.
[183] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks," in *Proc. IEEE Int. Conf. Data Mining*, 2016, pp. 221–230.
[184] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 177–187.
[185] F. Poursafaei, S. Huang, K. Pelrine, and R. Rabbany, "Towards better evaluation for dynamic link prediction," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 32928–32941.

ZhengZhao Feng is currently working toward the graduate degree majoring in software engineering with Zhejiang University. His research focuses on dynamic graph neural network models and frameworks.

Rui Wang received the PhD degree from the University of Science and Technology of China (USTC) in 2021 and now is a ZJU100 research fellow with the School of Software Engineering, Zhejiang University (ZJU). Her research interests lie in graph computing and storage systems, graph learning frameworks, machine learning systems, etc.

TianXing Wang is currently working toward the graduate degree majoring in software engineering with Zhejiang University. His research focuses on both graph neural network algorithms and systems.

Mingli Song received the PhD degree in computer science from Zhejiang University, China, in 2006. He is currently a professor with the Microsoft Visual Perception Laboratory, Zhejiang University. His research interests include face modeling and facial expression analysis. He received the Microsoft Research Fellowship, in 2004.

Sai Wu received the PhD degree from the National University of Singapore (NUS) in 2011 and now is a professor with the College of Computer Science, Zhejiang University. His research interests include distributed databases, AI for databases, and native AI database systems. He has won the best paper awards with VLDB 2014 and SIGMOD 2023. He has served as a Program Committee member for VLDB, ICDE, SIGMOD, and KDD.

Shuibing He (Member, IEEE) received the PhD degree in computer science and technology from the Huazhong University of Science and Technology in 2009. He is now a ZJU100 young professor with the College of Computer Science and Technology, Zhejiang University, China. His research areas include Intelligent computing, high-performance computing, memory, and storage systems. He is a member of the ACM.