Published: 09 May 2025
Online AM: 12 April 2025
Accepted: 08 March 2025
Revised: 07 November 2024
Received: 23 February 2024

RESEARCH-ARTICLE

DyExplainer: Self-explainable Dynamic Graph Neural Network with Sparse Attentions

TIANCHUN WANG, Pennsylvania State University, University Park, PA, United States

DONGSHENG LUO, Florida International University, Miami, FL, United States

WEI CHENG, NEC Laboratories America, Inc., Princeton, NJ, United States

HAIFENG CHEN, NEC Laboratories America, Inc., Princeton, NJ, United States

XIANG ZHANG, Pennsylvania State University, University Park, PA, United States

DyExplainer: Self-explainable Dynamic Graph Neural Network with Sparse Attentions

TIANCHUN WANG, The Pennsylvania State University, University Park, Pennsylvania, USA
DONGSHENG LUO, Florida International University, Miami, Florida, USA
WEI CHENG and HAIFENG CHEN, NEC Laboratories America, Princeton, New Jersey, USA
XIANG ZHANG, The Pennsylvania State University, University Park, Pennsylvania, USA

Graph Neural Networks (GNNs) resurge as a trending research subject owing to their impressive ability to capture representations from graph-structured data. However, the black-box nature of GNNs presents a significant challenge in terms of comprehending and trusting these models, thereby limiting their practical applications in mission-critical scenarios. Although there has been substantial progress in the field of explaining GNNs in recent years, the majority of these studies are centered on static graphs, leaving the explanation of dynamic GNNs less explored. Dynamic GNNs, with their ever-evolving graph structures, pose a unique challenge and require additional efforts to effectively capture temporal dependencies and structural relationships. To address this challenge, we present DyExplainer, a novel approach to explaining dynamic GNNs on the fly. DyExplainer trains a dynamic GNN backbone to extract representations of the graph at each snapshot, while simultaneously exploring structural relationships and temporal dependencies through a sparse attention technique. To preserve the desired properties of the explanation, such as structural consistency and temporal continuity, we augment our approach with contrastive learning techniques to provide *a priori*-guided regularization. To model longer-term temporal dependencies, we develop a buffer-based live-updating scheme for training. The results of our extensive experiments on various datasets demonstrate the superiority of DyExplainer, not only providing faithful explainability of the model predictions but also significantly improving the model prediction accuracy, as evidenced in the link prediction task.

CCS Concepts: • **Theory of computation → Dynamic graph algorithms**;

Additional Key Words and Phrases: dynamic graph neural networks, explanability, attentions

Associate Editor: Mahsa Salehi

ACM Reference format:
Tianchun Wang, Dongsheng Luo, Wei Cheng, Haifeng Chen, and Xiang Zhang. 2025. DyExplainer: Self-explainable Dynamic Graph Neural Network with Sparse Attentions. *ACM Trans. Knowl. Discov. Data.* 19, 4, Article 92 (May 2025), 21 pages.
https://doi.org/10.1145/3729173

Authors' Contact Information: Tianchun Wang (corresponding author), The Pennsylvania State University, University Park, Pennsylvania, USA; e-mail: tkw5356@psu.edu; Dongsheng Luo, Florida International University, Miami, Florida, USA; e-mail: dluo@fiu.edu; Wei Cheng, NEC Laboratories America, Princeton, New Jersey, USA; e-mail: weicheng@nec-labs.com; Haifeng Chen, NEC Laboratories America, Princeton, New Jersey, USA; e-mail: haifeng@nec-labs.com; Xiang Zhang, The Pennsylvania State University, University Park, Pennsylvania, USA; e-mail: xzz89@psu.edu.

ACM 1556-472X/2025/5-ART92
https://doi.org/10.1145/3729173

1 Introduction

The advent of **Graph Neural Networks (GNNs)** has caused a veritable revolution in the field and has been embraced with great enthusiasm due to their demonstrated efficacy in a variety of applications, ranging from node classification [16] and link prediction [70], to graph clustering [24, 56] and recommender systems [58]. However, these models are usually treated as black boxes, and their predictions lack understanding and explanations, thus preventing them to produce reliable solutions. Therefore, the study of the explainability of GNNs is in need. Recent research on GNN explainability mainly focuses on explaining the model predictions. However, the labyrinthine nature of GNNs has often resulted in their predictions being shrouded in mystery and lacking transparency. Consequently, the trustworthiness of their solutions is frequently called into question. To rectify this, the exploration and understanding of the explainability of GNNs have become imperative. The current research landscape in this arena primarily centers on demystifying the underlying mechanisms of GNN predictions, utilizing methods such as gradient-based techniques [1, 42], mask-based approaches [29, 63, 68], and surrogate models [54] to shed light on the reasoning [23] behind them. These techniques endeavor to bring greater transparency to the predictions of GNNs and to provide a deeper insight into the thought process of the model.

The advancement in explainability in static GNNs has been substantial, yet the same cannot be said for dynamic GNNs. Despite this, dynamic GNNs have gained widespread use in real-world applications, as they are capable of handling graphs that change over time. Elevating the level of explainability in dynamic GNNs is of utmost importance as it can foster greater human trust in the model's predictions. However, the dynamic and evolving nature of graphs presents several challenges. First, learning in dynamic graphs involves taking into consideration not only the topology and node attributes at each time point but also the temporal dynamics, making it a formidable task to present explanations in a manner that is easily understood by humans. Second, the consecutive order of dynamic graphs imposes unique constraints on the continuity of explanations. Lastly, the underlying patterns in dynamic graphs may also evolve with changes in node features and topologies, making it a challenge to find a balance between the continuity and evolution of explanations.

To handle these challenges, in this article, we propose DyExplainer, a dynamic explainable mechanism for dynamically interpreting GNNs. The framework is in Figure 1. DyExplainer constitutes a high-level, generalizable explainer that imparts insightful and comprehensible explanations through the utilization of graph patterns, thereby providing a deeper understanding of predictions made by diverse dynamic GNNs. The proposed approach incorporates pooling operations [30, 64] on nodes, thereby yielding graph-level embeddings that are encoder-agnostic and afford a remarkable degree of flexibility to a broad spectrum of dynamic encoders serving as the backbone. Furthermore, the introduction of the explainer module into the encoder incurs only a minimal overhead, making it highly efficient for large-scale backbone networks. Specifically, DyExplainer trains a dynamic GNN model to produce node embeddings at each snapshot, where the explainer module comprises two attention components: structural attention and temporal attention. The former leverages the pivotal relationships between nodes within the graph at each snapshot to inform the node representations, while the latter accounts for the temporal dependencies between representations generated by long-term snapshots. Despite the proliferation of dynamic GNNs [43, 65], most of these approaches deduce the representation at each snapshot t solely based on the embedding at the previous snapshot $(t-1)$, thereby making it challenging to fully comprehend the intricate, long-term temporal dependencies between snapshots in real-world applications. For instance, social network connections between individuals, groups, and communities may be subject to change over time, influenced by a multitude of factors, such as geographical distance, personal

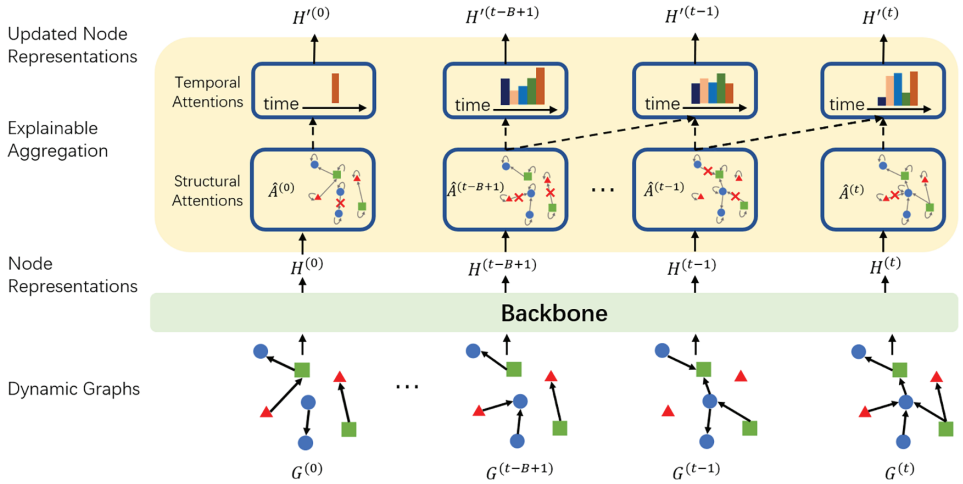


Fig. 1. DyExplainer for dynamic graphs. $\boldsymbol{H}^{(t)}$ is the node representation for snapshot $G^{(t)}$ after backbone module. $\boldsymbol{H'}^{(t)}$ is the updated node representation after explainable module. $\widehat{\boldsymbol{A}}^{(t)}$ is the structural attention for snapshot $G^{(t)}$.

life stage, and evolving interests. To encompass longer-term temporal dependencies, we have devised a buffer-based, live-updating scheme with temporal attention. Specifically, the temporal attention aggregates the node embeddings from the preceding B snapshots stored in a buffer. In this regard, prior works on dynamic GNNs are reduced to a special case of the DyExplainer, where $B = 1$. The proposed framework constitutes a generalization that encompasses all recent graph learning methods for dynamic graphs. For example, by relinquishing explainability and incorporating the Markov chain property for temporal evolution, our framework degenerates to ROLAND [65]. Additionally, by restricting temporal aggregations to only the final layer, our framework degenerates to a common approach in which a sequence model, such as GRU [6], is placed on top of GNNs [40, 57, 66].

The dual sparse attention components of DyExplainer serve a trifold function. First, they impart incisive explanations for the model's predictions in downstream tasks. Second, they serve as a sparse regularizer, enhancing the learning process. As research in the field of ranshomon set theory [44, 60] demonstrates, sparse and interpretable models possess a higher degree of robustness and better generalization performance, as outlined in Section 5.3. Lastly, the attention components allow for the augmentation of the approach with *a priori*-guided regularization, preserving the desirable properties of the explanation, such as structural consistency and temporal continuity. Structural consistency pertains to the consistency of node embeddings between connected nodes in a single snapshot, while temporal continuity enforces smoothness constraints on the temporal attention between closely spaced snapshots, guided by pre-established human priors. To achieve more lucid explanations, we employ the use of contrastive learning techniques, treating connected pairs as positive examples and unconnected pairs as negative examples for consistency regularization, and recent snapshots as positive examples and distant historical snapshots as negative examples for continuity regularization. Overall, our contributions are summarized as follows:

—We address the problem of explanability for dynamic GNNs and propose a general self-explainable method DyExplainer, which seamlessly integrates the modeling of both structural relationships and long-term dependencies via sparse attentions.

—DyExplainer is capable of providing real-time explanations for both the structural and temporal factors that influence the model's predictions. This innovative explaining module has been designed to be encoder-agnostic, thereby affording flexibility to a range of dynamic GNN backbones. Its implementation requires minimal overhead, as it only entails adding a lightweight parameterization to the encoder for its explanation modules. This makes DyExplainer highly efficient even for large backbone networks.
—We propose two contrastive regularizations to provide consistency and continuity explanations. Our approach to augmenting desired properties in the explanation is a fresh contribution to the field and may be of independent interest.
—Extensive experiments on various datasets demonstrate the superiority of DyExplainer, not only providing faithful explainability of the model predictions but also significantly improving the model prediction accuracy, as evidenced in the link prediction task.

2 Related Work

In this section, we review the literature in two groups: dynamic GNNs and explainability in GNNs.

2.1 Dynamic GNNs

The majority of current GNN models are optimized for static graphs, wherein the graph structure remains constant. These models undergo a single iteration on the graph and make predictions based on their unchanging structure [11, 16, 53]. However, real-world graphs tend to be dynamic and constantly evolving over time. This has generated increased interest in the development of dynamic GNNs, which can effectively deal with changing graph structures and capture the evolution of the graph. Below, we provide an overview of the key techniques employed in dynamic GNNs.

Several works have combined GNNs with recurrent models, such as GRU cells, to capture the evolution of the graph structure over time [22, 32, 69], which are referred to as recurrent GNNs. These models process a sequence of graph snapshots, where each snapshot represents the graph structure at a given timestep. The recurrent modules, such as RNNs or LSTMs, are integrated into the GNN architecture to preserve the temporal dependencies between graph states in dynamic graph sequences and to incorporate both graph structure and node features. The GNN component processes information from the graph structure, while the recurrent component manages the dynamic aspect of the data. Despite their potential, recurrent GNNs can be challenging to train, given a large number of parameters and difficulties in capturing long-term dependencies. Differently, DyExplainer leverages recurrent GNNs as its backbone, but updates dynamically, eliminating the need for computing all snapshots. Its temporal attention mechanism is designed to capture long-term dependencies.

Another group of approaches that are particularly relevant to this work is attention-based GNNs in dynamic graphs [45, 61, 74]. Attention mechanisms have been introduced into GNNs to dynamically focus on the significance of different graph structures or node features in dynamic graphs. This allows the network to concentrate on the most valuable aspects of the graph at various timesteps, leading to improved performance on tasks such as node classification and link prediction, and providing interpretable subgraph structures. However, these approaches lack the ability to ensure the continuity of attention edges for consecutive snapshots, making it difficult to detect essential structures in short-term snapshots. Additionally, they often face computational challenges for long-range snapshots. In contrast, DyExplainer calculates temporal attention in a buffer, alleviating computational burden. Additionally, its consistency regularization ensures that attention values are consistent with those of nearby snapshots.

2.2 Explainability in GNNs

The goal of explainability in GNNs is to provide transparency and accountability in the predictions of the models, especially when they are used in critical applications such as detecting the fraud [25] or helping the human to make medical diagnoses [27]. Recently, many works are proposed to explain the GNN predictions, with a focus on diverse facets of the models from various perspectives. Recently, some works in explainability of GNNs are emerging [67]. These methods can be grouped into two categories: *post hoc* methods and self-explainable methods.

The majority of the GNN explainers are *post hoc* methods, which identify key nodes, edges, and/or node features of the input instance to provide instance-level explanations for GNN models' predictions. SA [1] computes the gradient value as the importance score for each input feature. While it is easy to compute by the back-propagation, the results cannot accurately capture the importance of each feature due to the output changing minimally with respect to the input change, and a gradient value is hard to reflect the input contribution. CAM [42] maps the node features in the final layer to the input space to identify important nodes. However, the representation from the final layer of GNN may not reflect the node contribution because the feature distribution may change after mapping by a neural network. Another kind of method aims to provide instance-level explanations by a surrogate model that approximates the predictions of the original GNN model. GraphLime [14] provides a model-agnostic local explanation framework for the node classification task. It adopts the node feature and predicted labels in the K-hop neighbors of the predicted node and trains an HSIC Lasso. For instance, PGM-Explainer [54] identifies crucial graph components and generates an explanation in the form of a probabilistic graphical model as a surrogate to approximate the prediction. The PGM-Explainer can be performed to explain node and graph classification tasks. More recently, a group of methods study the output variations of GNN with respect to the input perturbations by learning a mask that selects important features to explain the given prediction. As a typical method, GNNExplainer [63] takes a trained GNN and its predictions as inputs to provide explanations for a given instance, e.g., a node or a graph. The explanation includes a compact subgraph structure and a small subset of node features that are crucial in GNN's prediction for the target instance. However, the explanation provided by GNNExplainer is limited to a single instance, making GNNExplainer difficult to be applied in the inductive setting because the explanations are hard to generalize to other unexplained nodes. PGExplainer [29] is proposed to provide a global understanding of predictions made by GNNs. It models the underlying structure as edge distributions where the explanatory graph is sampled. To explain the predictions of multiple instances, the generation process in PGExplainer is parameterized by a neural network. Similar to the PGExplainer, GraphMask [46] trains a classifier to predict whether an edge can be dropped without affecting the original predictions. As a *post hoc* method, GraphMask obtains an edge mask for each GNN layer. SubgraphX [68] explains the GNN predictions as an efficient exploration of different subgraphs with Monte Carlo tree search. It adopts Shapley values as a measure of subgraph importance that also captures the interactions among different subgraphs.

The methods mentioned above primarily focus on *post hoc* explanations for trained GNN models, often requiring an additional explainer to generate explanations for the model. In contrast, there are few works on self-explainable GNNs, which aim to provide predictions and explanations simultaneously. ProtGNN [71] combines prototype learning with GNNs, and the explanations are naturally derived from the case-based reasoning process during the graph classification task. SE-GNN [7] finds k-nearest labeled nodes for each unlabeled node to give explainable node classification. MSE-GNN [41] adopts a two-stage self-explaining structure and a meta-training framework based on meta learning.

Table 1. Notations

Notation	Definition
$G^{(t)}, \mathcal{V}, \mathcal{E}^{(t)}$	Graph at snapshot t; set of nodes of dynamic graph; set of edges at snapshot t.
$H^{(t)}$	Node embedding of the graph after the encoder backbone at the snapshot t.
$\widehat{\boldsymbol{A}}^{(t)}$	Structural attention matrix at the snapshot t.
$\widetilde{\boldsymbol{H}}^{(t)}$	Node embedding after the structural attention at the snapshot t.
$\boldsymbol{H}^{'(t)}$	Node embedding after the temporal attention at the snapshot t.
$\widetilde{\boldsymbol{A}}^{(i)}$	Temporal attention matrix if node i.
$\boldsymbol{M}_t$	Temporal mask used in the buffer.
$\boldsymbol{W}^{(s)}, \boldsymbol{W}$	Weight matrix for linear transformation in structural attention and temporal attention, respectively.
$\boldsymbol{a}^{(s)}, \boldsymbol{a}$	Weight vector in the single-layer network in structural attention and temporal attention, respectively.
B	Buffer size.
F	Feature dimension of node embedding after the backbone.
F', K	Feature dimension of node embedding after the structural attention and temporal attention, respectively.
α, β	Tradeoff parameters for the regularizations
τ	Temperature in Gumbel-softmax.

Current explanation methods for GNNs are limited to static graphs, hindering their application in dynamic scenarios. The explanation of dynamic GNNs is an under-studied area. A recent work [59] attempted to provide explanations by exploring backward relevance, but it was limited to a specific model TGCN. There are distinct challenges in explaining dynamic GNNs in general. First, existing explanation methods mainly focus on identifying the important parts of the data in relation to the GNN predictions. However, dynamic GNNs make predictions for each snapshot, making it difficult for an explainer to provide explanations for a particular prediction as it depends on all previous snapshots. Second, as discussed in Section 2.1, there are many types of dynamic GNNs, making it challenging to provide a generalizable solution. DyExplainer, on the other hand, provides self-explainable scheme for all dynamic GNNs, overcoming these limitations.

3 Notations and Problem Definition

We formulate a dynamic graph as a series of T attributed graphs as $\mathcal{G} = (G^{(1)}, ..., G^{(T)})$, where each graph $G^{(t)} = (\mathcal{V}, \mathcal{E}^{(t)}, \boldsymbol{X}^{(t)})$ is a snapshot at timestep t. $\mathcal{V} = \{v_1, v2, ...v_N\}$ is the set of N nodes shared by all snapshots and $\mathcal{E}^{(t)} \in \mathcal{V} \times \mathcal{V}$ is the set of edges of a graph $G^{(t)}$. $\boldsymbol{X}^{(t)} \in \mathbb{R}^{N \times D}$ is the node feature matrix at snapshot t, where D is the number of dimensions of node features. The topology and node features of each snapshot are dynamically changing over time. We aim to learn an explainable dynamic GNN model f from the long-term snapshots defined as:

$$\widehat{\boldsymbol{A}}^{(t-B+1)}, ..., \widehat{\boldsymbol{A}}^{(t)}, \boldsymbol{H'}^{(t+1)} \leftarrow f\left(G^{(t-B+1)}, ..., G^{(t)}\right), \tag{1}$$

where $\widehat{\boldsymbol{A}}^{(t-B+1)}, ..., \widehat{\boldsymbol{A}}^{(t)}$ are explanations we are interested and $\boldsymbol{H'}^{(t+1)}$ is the future node representation predicted for the downstream tasks. In our study, we examine its application for link prediction. It's worth noting that our framework can also easily support other downstream tasks. Important notations are summarized in Table 1.

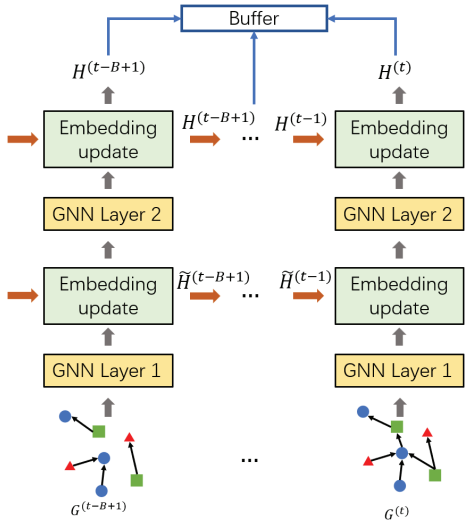


Fig. 2. Backbone encoder architecture.

4 Method

4.1 Encoder Backbone

There are different approaches to designing dynamic GNNs. As a plug-and-use framework, the proposed DyExplainer is flexible to the choice of dynamic GNNs as the backbone. In this work, we adopt a state-of-the-art method ROLAND [65] as the backbone due to its powerful expressiveness and impressive empirical performances in real-life datasets. Specifically, it hierarchically updates the node embedding to obtain the $\boldsymbol{H}^{(t)}$ at snapshot t. With ROLAND as the backbone, the architecture of the encoder is shown in Figure 2. We put the node embedding inferred by the backbone to a buffer of size B for the learning of our explainer module. Formally, we denote the node embeddings in the buffer as $\{\boldsymbol{H}^{(t-B+1)}, ..., \boldsymbol{H}^{(t)}\}$.

4.2 Explainable Aggregations

The node embedding at snapshot t is denoted by $\boldsymbol{H}^{(t)} = \{\boldsymbol{h}_1, ..., \boldsymbol{h}_N\}$, $\boldsymbol{h}_i \in \mathbb{R}^F$, where N is the number of nodes, and F is the feature dimension of the node embedding.

4.2.1 Structural Aggregation. On dynamic graphs, each snapshot has its unique topology information. An effective explanation, therefore, should highlight the crucial structural components at a given snapshot that significantly contribute to the model's prediction. Drawing inspiration from graph attention mechanisms, which assign varying levels of importance to graph edges that can be used as explanations [34, 48, 53], we propose a structural attention mechanism to aggregate weighted neighborhoods at each timestep. Formally, we have

$$\omega_{ij}^{(s,t)} = \text{LeakyReLU}\left(\boldsymbol{a}^{(s)T}\left[\boldsymbol{W}^{(s)}\boldsymbol{h}_i^{(t)} || \boldsymbol{W}^{(s)}\boldsymbol{h}_j^{(t)}\right]\right), \tag{2}$$

$$\widehat{\boldsymbol{A}}_{ij}^{(t)} = \text{softmax}_j\left(\omega_{ij}^{(s,t)}\right) = \frac{\exp\left(\omega_{ij}^{(s,t)}\right)}{\sum_{k\in\boldsymbol{N}_i^{(t)}}\exp\left(\omega_{ik}^{(s,t)}\right)}, \tag{3}$$

where $\widehat{\boldsymbol{A}}^{(t)}$ is the structural attention at timestep t, $\boldsymbol{W}^{(s)}$ is a weight matrix in a shared linear transformation, $\boldsymbol{a}^{(s)}$ is the weight vector in the single-layer network, $\mathcal{N}_i^{(t)}$ is the set of neighbors of node i at timestep t, and || indicates the concatenation operation.

However, the utilization of weights in traditional attention models may pose a challenge in complex dynamic graph environments, particularly with regard to explanation. Explanations in these settings are often derived by imposing a threshold and disregarding insignificant attention weights. This approach, however, fails to account for the cumulative impact of numerous small but non-zero weights, which can be substantial. Moreover, the non-exclusivity of attention weights raises questions about their accuracy in reflecting the true underlying importance [53]. To address this issue and equip structural attention with better explainability, we design a hard attention mechanism to alleviate the effects of small attention coefficients. The basic idea uses a prior work on differentiable sampling [15, 31], which states that the random variable

$$e = \sigma\Big(\big(\log\epsilon - \log\left(1-\epsilon\right) + \omega\big)/\tau\Big) \quad \text{where} \quad \epsilon \sim \mathcal{U}(0,1), \tag{4}$$

where $\sigma(\cdot)$ is the sigmoid function and τ is the temperature controlling the approximation. Equation (4) follows a distribution that converges to a Bernoulli distribution with success probability $p = (1+e^{-\omega})^{-1}$ as $\tau > 0$ tends to zero. Hence, if we parameterize ω and specify that the presence of an edge between a pair of nodes has probability p, then using e computed from Equation (4) to fill the corresponding entry of $\widehat{\boldsymbol{A}}$ will produce a matrix $\widehat{\boldsymbol{A}}$ that is close to binary. We use this matrix as the hard attention with the hope of obtaining a better explanation due to the dropping of small attention weights. Moreover, because Equation (4) is differentiable with respect to ω, we can train the parameters of ω like in a usual gradient-based training. In the structural attention, we have the parameterized $\omega_{ij}^{(s,t)}$ in Equation (2), thus we get

$$\widetilde{e}_{ij}^{(s,t)} = \sigma\Big(\big(\log\epsilon - \log\left(1-\epsilon\right) + \omega_{ij}^{(s,t)}\big)/\tau\Big) \quad \text{where} \quad \epsilon \sim \mathcal{U}(0,1),$$

which returns an approximate Bernoulli sample for the edge (i, j). When τ is not sufficiently close to zero, this sample may not be close enough to binary, and in particular, it is strictly non-zero. The rationality of such an approximation is that with temperature $\tau > 0$, the gradient $\frac{\partial \widetilde{e}_{ij}^{(s,t)}}{\partial \omega_{ij}^{(s,t)}}$ is well-defined. The output of the binary concrete distribution is in the range of (0,1). To further alleviate the effects of small values by encouraging them to be exactly zero, we propose a "stretching and clipping" technique in the hard attention mechanism. To explicitly zero out an edge, we follow [28] and introduce two parameters, $\gamma < 0$ and $\xi > 1$, to remove small values of $\widetilde{e}_{ij}^{(s,t)}$ given by

$$\widehat{\boldsymbol{A}}_{ij}^{(t)} = \min\Big(1, \max\left(e_{ij}^{(s,t)}, 0\right)\Big) \quad \text{where} \quad e_{ij}^{(s,t)} = \widetilde{e}_{ij}^{(s,t)}(\xi-\gamma) + \gamma.$$

The structural attention $\widehat{\boldsymbol{A}}^{(t)}$ does not insert new edges to the graph (i.e., when $(i,j) \notin \mathcal{E}^{(t)}$, $\widehat{\boldsymbol{A}}_{ij}^{(t)} = 0$), but only removes (denoises) some edges (i, j) originally in $\mathcal{E}^{(t)}$. Then, we obtain the node embedding $\widetilde{\boldsymbol{H}}^{(t)} \in \mathbb{R}^{N\times F'}$ at the timestep t after the structural aggregation with each row given by

$$\widetilde{\boldsymbol{h}}_i^{(t)} = \sigma\Big(\sum_{j\in\mathcal{N}_i^{(t)}} \widehat{\boldsymbol{A}}_{ij}^{(t)} \boldsymbol{W}^{(s)} \boldsymbol{h}_j^{(t)}\Big). \tag{5}$$

(a) Local (b) Random

Fig. 3. Examples of temporal mask in temporal aggregation. "Local" means considering continuous snapshots before the current timestep. "Random" means randomly picking historical snapshots before the current timestep. The value is 1 in green boxes and 0 otherwise.

4.2.2 Temporal Aggregation. In light of the dynamic and evolving nature of node features and inter-node relationships over time, temporal dependency holds paramount importance in the modeling of dynamic graphs. Existing methods either adopt RNN architectures, such as GRU [6] and LSTM [12], or assume the Markow chain property [65] to capture the temporal dependencies in dynamic graphs. As shown in [52], despite their utility, these methods are insufficient in capturing long-range dependencies, thereby hindering their ability to generalize and model previously unseen graphs. To overcome this limitation, we propose a solution that leverages an attention-based temporal aggregation mechanism to adaptively integrate node embeddings from distant snapshots. This is achieved through the utilization of a buffer-dependent temporal mask, which serves as a temporal topology to guide the aggregation process. The temporal mask examples are shown in Figure 3.

In Equation (5), the structural attention provides the node embedding $\widetilde{\boldsymbol{H}}^{(t)} \in \mathbb{R}^{N\times F'}$ for each snapshot t. Therefore, we have a set of node embedding $\{\widetilde{\boldsymbol{H}}^{(t-B+1)}, ..., \widetilde{\boldsymbol{H}}^{(t)}\}$. Concatenating them to a 3D tensor and take transpose, for each node i, we have a buffer-dependent node embedding $\widehat{\boldsymbol{H}}^{(i)} \in \mathbb{R}^{B\times F'}$, $\widehat{\boldsymbol{H}}^{(i)} = \{\widehat{\boldsymbol{h}}_{t-B+1}, ..., \widehat{\boldsymbol{h}}_t\}$, $\widehat{\boldsymbol{h}}_t \in \mathbb{R}^{F'}$. We propose the temporal attention given by

$$\omega^{(i)}_{t_k,t_j} = \text{LeakyReLU}\Big(\boldsymbol{a}^T\Big[\boldsymbol{W}\boldsymbol{h}^{(i)}_{t_k} || \boldsymbol{W}\boldsymbol{h}^{(i)}_{t_j}\Big]\Big), \tag{6}$$

$$\widetilde{\boldsymbol{A}}^{(i)}_{t_k,t_j} = \text{softmax}_{t_j}\Big(\omega^{(i)}_{t_k,t_j}\Big) = \frac{\exp\Big(\omega^{(i)}_{t_k,t_j}\Big)}{\sum_{t_p\in\mathcal{M}_{t_k}}\exp\Big(\omega^{(s,t)}_{t_k,t_p}\Big)}, \tag{7}$$

where $\widetilde{\boldsymbol{A}}^{(i)}$ is the temporal attention for node i, $\boldsymbol{W}$ is a weight matrix for linear transformation, $\boldsymbol{a}$ is a weight vector for single-layer network, $\mathcal{M}_{t_k}$ is the timesteps that has element 1 in the temporal mask. The values in $\widetilde{\boldsymbol{A}}^{(i)} \in \mathbb{R}^{B\times B}$ indicate the importance of relations between the embedding for node i at the past snapshots. We compute Equations (6) and (7) in batch for acceleration due to the graphs usually have large numbers of nodes. Then, after the temporal attention, we obtain the node embedding $\boldsymbol{H}^{'(i)} \in \mathbb{R}^{B\times K}$ for each node i for all the B snapshots in the buffer, with each row given by

$$\boldsymbol{h}^{'(i)}_{t_k} = \sigma\Big(\sum_{t_j\in\boldsymbol{M}_{t_k}}\widetilde{\boldsymbol{A}}^{(i)}_{t_k,t_j}\boldsymbol{W}\boldsymbol{h}^{(i)}_{t_j}\Big). \tag{8}$$

Therefore, we have the embedding of node i at time t is $\boldsymbol{h}^{'(i)}_t \in \mathbb{R}^K$, and the embedding of all nodes results in $\boldsymbol{H}^{'(t)} \in \mathbb{R}^{N\times K}$, $\boldsymbol{H}^{'(t)} = \{\boldsymbol{h}^{'(0)}_t, ..., \boldsymbol{h}^{'(N-1)}_t\}$.

4.3 Regularizations

The framework of DyExplainer is flexible with various regularization terms to preserve desired properties on the explanation. Inspired by the graph contrastive learning that makes the node representations more discriminative to capture different types of node-level similarity, we propose a structural consistency and a continuity consistency. We now discuss the regularization terms as well as their principles.

4.3.1 Consistency Regularization. Inspired by the homophily nature of graph-structured data [33], we propose a topology-wise regularization to encourage consistent explanations of the connected nodes in a graph. Specifically, on the graph $G^{(t)} = \{\mathcal{V}, \mathcal{E}^{(t)}\}$, for a node $i \in \mathcal{V}$, we sample a positive pair (i, p), the edge $e_{i,p} \in \mathcal{E}^{(t)}$. We sample unconnected pairs (i, j) such that $e_{i,j} \notin \mathcal{E}^{(t)}$ to form a set of non-negative samples $\bar{\mathcal{N}}_i$, then we propose the consistency regularization for the structural attention $\widehat{\boldsymbol{A}}^{(t)}$ as

$$\mathcal{L}_{cons} = -\log \frac{\exp\left(\text{sim}\left(\widehat{\boldsymbol{A}}_i^{(t)}, \widehat{\boldsymbol{A}}_p^{(t)}\right)\right)}{\exp\left(\text{sim}\left(\widehat{\boldsymbol{A}}_i^{(t)}, \widehat{\boldsymbol{A}}_p^{(t)}\right)\right) + \sum_{j \in \bar{\mathcal{N}}_i} \exp\left(\text{sim}\left(\widehat{\boldsymbol{A}}_i^{(t)}, \widehat{\boldsymbol{A}}_j^{(t)}\right)\right)}. \tag{9}$$

Note that the computation of Equation (9) is very time-consuming for graphs with large numbers of nodes and edges. In practice, we select some anchors to compute the $\mathcal{L}_{cons}$.

4.3.2 Continuity Regularization. As suggested in [35], preserving continuity ensures the robustness of explanations that small variations applied to the input, for which the model prediction is nearly unchanged, will not lead to large differences in the explanation. In addition, continuity benefits generalizability beyond a particular input instance. Based on the practical principle, in dynamic graphs, we aim to maintain a consistent explanation of snapshots even as the graph structure evolves. Inspired by the idea in [51] that two close subsequences are considered as a positive pair while the ones with large distances are the negatives, we propose a continuity regularization.

For a snapshot G^t, a positive pair (G^t, G^p) is sampled from the sliding window in the temporal mask. The set of non-negative samples $\bar{\mathcal{N}}_t$ is historical snapshots that are not in the buffer. Then, the continuity regularization for each node i is given by

$$\mathcal{L}_{cont} = -\log \frac{\exp\left(\text{sim}\left(\widetilde{\boldsymbol{A}}_i^{(t)}, \widetilde{\boldsymbol{A}}_i^{(p)}\right)\right)}{\exp\left(\text{sim}\left(\widetilde{\boldsymbol{A}}_i^{(t)}, \widetilde{\boldsymbol{A}}_i^{(p)}\right)\right) + \sum_{k \in \bar{\mathcal{N}}_t} \exp\left(\text{sim}\left(\widetilde{\boldsymbol{A}}_i^{(t)}, \widetilde{\boldsymbol{A}}_i^{(k)}\right)\right)}, \tag{10}$$

where $\widetilde{\boldsymbol{A}}_i^{(t)} \in \mathbb{R}^{B \times B}$ is the temporal attention of node i on G^t. To compute Equation (10) for all of the nodes instead of one-by-one, we form a block diagonal matrix with each diagonal block be the attention corresponding to each node, say $\widetilde{\boldsymbol{A}}_{\text{block}}^{(t)} = \text{diag}\left(\widetilde{\boldsymbol{A}}_0^{(t)}, ..., \widetilde{\boldsymbol{A}}_N^{(t)}\right)$.

4.4 Buffer-based Live-update

After we obtain the node embedding $\boldsymbol{H}^{'(t)} \in \mathbb{R}^{N \times K}$ for the current snapshot, DyExplainer uses an MLP to predict the probability of a future edge from node i to j. We compute a cross-entropy loss $\mathcal{L}_{ce}$ between the predictions and the edge labels at the future snapshot. After all, we have the objective function

$$\mathcal{L} = (1 - \alpha)\mathcal{L}_{ce} + \alpha(\mathcal{L}_{cons} + \mathcal{L}_{cont}), \tag{11}$$

Algorithm 1: Buffer-based Live-update Algorithm

Input: Dynamic graphs $\mathcal{G} = (G^{(1)}, ..., G^{(T)})$, link prediction labels $y_1, ...y_T$, number of snapshots T, maximum fine-tuning epoch E, maximum buffer size B, trade-off parameters α and β, the DyExplainer model.
Initialize the node state $H^{(0)}$ and an empty queue as a buffer with size 0;
for $t = 1, ..., T - 1$ **do**
  Train and update the backbone module based on y_t with early stopping and get $H^{(t)}$;
  if Buffer size < B **then**
    Insert $\boldsymbol{H}^{(t)}$ into the buffer;
  else
    Delete $\boldsymbol{H}^{(t-B)}$ and insert $\boldsymbol{H}^{(t)}$;
  end if
  for $e = 1, ..., E$ **do**
    Input the node embedding stored in the buffer to the structural attention and get $\widetilde{\boldsymbol{H}}^{(t)}$, as described in Section 4.2.1;
    Compute $\mathcal{L}_{cons}$ in equation 9;
    Input the structural node embedding to the temporal attention and get $\boldsymbol{H}^{'(t)}$, as described in Section 4.2.2;
    Compute $\mathcal{L}_{cont}$ in equation 10;
    Compute $\mathcal{L}$ in equation 11;
    Train and Update the attention modules based on y_t.
  end for
end for

where α is the tradeoff parameter. Inspired by ROLAND [65], we develop a buffer-based live-update algorithm to train the model. The key idea is to balance the efficiency and the aggregation from historical embeddings. Specifically, we update the backbone lively and fine-tuning the attention modules with the node embedding from the buffer. Note that the backbone is updated based on cross-entropy loss because without attention module, the terms $\mathcal{L}_{cons}$ and $\mathcal{L}_{cont}$ are zeros. We provide the details in Algorithm 1.

5 Experiments

In evaluating the performance of the DyExplainer, we will compare it against state-of-the-art baselines. Our findings demonstrate the significant improvement in model generalization for link prediction achieved by the DyExplainer. Furthermore, we quantitatively validate the accuracy of its explanations. We also delve deeper with ablation studies and a case study, offering a deeper understanding of the proposed method.

5.1 Experimental Setup

5.1.1 Datasets. The experiments are performed on six widely used datasets in the following list. The dataset (1) AS-733 is an autonomous systems dataset of traffic flows among routers comprising the Internet [20]. (2) Reddit-Title and (3) Reddit-Body are networks of subreddit-to-subreddit hyperlinks extracted from posts. The posts contain hyperlinks that connect one subreddit to another. The edge label shows if the source post expresses negativity toward the target post [17]. (4) UCI-Message is composed of private communications exchanged on an online social network system among students [36]. The datasets Bitcoin-OTC and Bitcoin-Alpha consist of "who-trusts-whom" networks of individuals who engage in trading on these platforms [18, 19]. For statistics of the datasets, see Table 2.

5.1.2 Baselines. We compare DyExplainer with both link prediction methods and explainable methods. For link prediction methods, we use seven state-of-the-art dynamic GNNs. The (1) EvolveGCN-H and (2) EvolveGCN-O models employ an RNN to dynamically adapt the weights of internal GNNs, enabling the GNN to change during testing [37]. (3) T-GCN integrates a GNN into the GRU cell by replacing the linear transformations in GRU with graph convolution operators [73]. The (4) GCRN-GRU and (5) GCRN-LSTM methods are widely adopted baselines that are generalized

Table 2. Dataset Statistics

Dataset	# Nodes	# Edges	Range	# Snapshot Frequency	# Snapshots
AS-733	7,716	11,965,533	8 November 1997–2 January 2000	Daily	733
Reddit-Title	54,075	571,927	31 December 2013–30 April 2017	Weekly	178
Reddit-Body	35,776	286,561	31 December 2013–30 April 2017	Weekly	178
UCI-Message	1,899	59,835	15 April 2004–26 October 2004	Weekly	29
Bitcoin-OTC	5,881	35,592	8 November 2010–24 January 2016	Weekly	279
Bitcoin-Alpha	3,783	24,186	7 November 2010–21 January 2016	Weekly	274

to capture temporal information by incorporating either a GRU or an LSTM layer. GCRN uses a ChebNet [8] for spatial information and separate GNNs to compute different gates of RNNs. The (6) GCRN-Baseline first builds node features using a Chebyshev spectral graph convolution layer to capture spatial information, then feeds these features into an LSTM cell to extract temporal information [47]. The (7) ROLAND views the node embeddings at different layers in the GNNs as hierarchical node states, which it updates recurrently over time. It integrates advanced design features from static GNNs and enables lively updating. Throughout our experiments, we use the GRU-based ROLAND which is shown to perform better than the others [65].

To evaluate the effectiveness of explainability, we compare DyExplainer with GNNExplainer [63] and a gradient-based method (Grad). (1) GNNExplainer is a *post hoc* state-of-the-art method providing explanations for every single instance. (2). Grad learns weights of edges by computing gradients of the model's objective function w.r.t. the adjacency matrix.

5.1.3 Metrics. For measuring link prediction performance, we use the standard **Mean Reciprocal Rank (MRR)**. For evaluating the faithfulness of explainability, we mainly use the Fidelity score of probability [68]. We let $\mathcal{G}^t = \{\mathcal{V}_t, \mathcal{E}_t\}$ be the graph at the snapshot t, with $\mathcal{V}_t$ a set of vertices and $\mathcal{E}_t$ a set of edges. For snapshot t, there is a set of edges $\mathcal{E}'_t$ needed to make predictions. After training the DyExplainer, we obtain an explanation mask $\boldsymbol{M}_t \in \{0,1\}^{n\times n}$ for the snapshot $\mathcal{G}^t$, with each element 0 or 1 to indicate whether the corresponding edge is identified as important. According to $\boldsymbol{M}_t$ we obtain the important edges in $\mathcal{G}^t$ to create a new graph $\hat{\mathcal{G}}^t$. The Fidelity score is computed as

$$Fidelity = \frac{1}{|\mathcal{E}'_t|}\sum_{i=1}^{|\mathcal{E}'_t|}\left|f(\mathcal{G}^t)_{e'_i} - f(\hat{\mathcal{G}}^t)_{e'_i}\right|, \tag{12}$$

where $|\mathcal{E}'_t|$ is the number of edges need to predict, $f(\mathcal{G}^t)_{e'_i}$ means the predicted probability of edge $e'_i \in \mathcal{E}'_t$. We follow [42, 67] to compute the Fidelity scores at different sparsity given by

$$Sparsity = 1 - \frac{|\boldsymbol{M}_t|}{|\mathcal{E}_t|}, \tag{13}$$

where $|\boldsymbol{M}_t|$ is the number of important edges identified in $\boldsymbol{M}_t$ and $|\mathcal{E}_t|$ means the number of edges in $\mathcal{G}_t$.

5.1.4 Implementation Details. To ensure a fair comparison, all methods were trained through live updating as outlined in [65]. The evaluation is supposed to happen at each snapshot; however, models trained on streaming data are not stabilized in the early stages. In the real world, we usually have more interest in the predictions of more recent snapshots. Therefore, we report the average of MRR of the most recent 60% snapshots. The hyper-parameter space of the DyExplainer model is

Table 3. Comparison of MRR for the DyExplainer and the Baselines

	AS-733	Reddit-Title	Reddit-Body	UCI-Message	Bitcoin-OTC	Bitcoin-Alpha
EvolveGCN-H	0.263 ± 0.098	0.156 ± 0.121	0.072 ± 0.010	0.055 ± 0.011	0.081 ± 0.025	0.054 ± 0.019
EvolveGCN-O	0.180 ± 0.104	0.015 ± 0.019	0.093 ± 0.022	0.028 ± 0.005	0.018 ± 0.008	0.005 ± 0.006
GCRN-GRU	**0.337 ± 0.001**	0.328 ± 0.005	0.204 ± 0.005	0.095 ± 0.013	0.163 ± 0.005	0.143 ± 0.004
GCRN-LSTM	0.335 ± 0.001	0.343 ± 0.006	0.209 ± 0.003	**0.107 ± 0.004**	0.172 ± 0.013	0.146 ± 0.008
GCRN-Baseline	0.321 ± 0.002	0.342 ± 0.004	0.202 ± 0.002	0.090 ± 0.011	0.176 ± 0.005	**0.152 ± 0.005**
TGCN	0.335 ± 0.001	0.382 ± 0.005	0.234 ± 0.004	0.080 ± 0.015	0.080 ± 0.006	0.060 ± 0.014
ROLAND	0.330 ± 0.004	**0.384 ± 0.013**	**0.342 ± 0.008**	0.090 ± 0.010	**0.189 ± 0.008**	0.147 ± 0.006
DyExplainer	**0.341 ± 0.000**	**0.383 ± 0.002**	**0.335 ± 0.010**	**0.109 ± 0.004**	**0.194 ± 0.002**	**0.164 ± 0.002**

SDs are obtained by repeating each model training five times. For each dataset, the two best cases are boldface.

similar to the hyper-parameter space of the underlying static GNN layers. For all methods, the node state hidden dimensions are set to 128 with GNN layers featuring skip connection, sum aggregation, and batch normalization. The training of DyExplainer in each timestep has two parts, the updating of ROLAND backbone, and the updating of explainable modules. Training DyExplainer at each timestep consists of two parts: updating the ROLAND backbone and updating the explainable modules. The ROLAND backbone is trained for a maximum of 100 epochs at each timestep with early stopping, followed by fine-tuning the explainable module for an additional four epochs in all experiments. During training, the batch size is set to 32. We use Adam for optimization with the initial learning rate set to 0.003 and weight decay set to 0.0005, for both the ROLAND backbone and the explainable modules of our method. In the attention modules, the slope in LeakyReLU is set to 0.2. We follow the practice in [15] to adopt the exponential decay strategy by starting the training with a high temperature of 1.0 and annealing to a small value of 0.1. For each dataset, we adhere to the parameter settings used for the ROLAND backbone in [65]. Specifically, we set the hidden dimension for the structural and temporal attention modules to 8, the buffer size for the explainable module to 3, and the regularization tradeoff parameter α within the range (0,1).

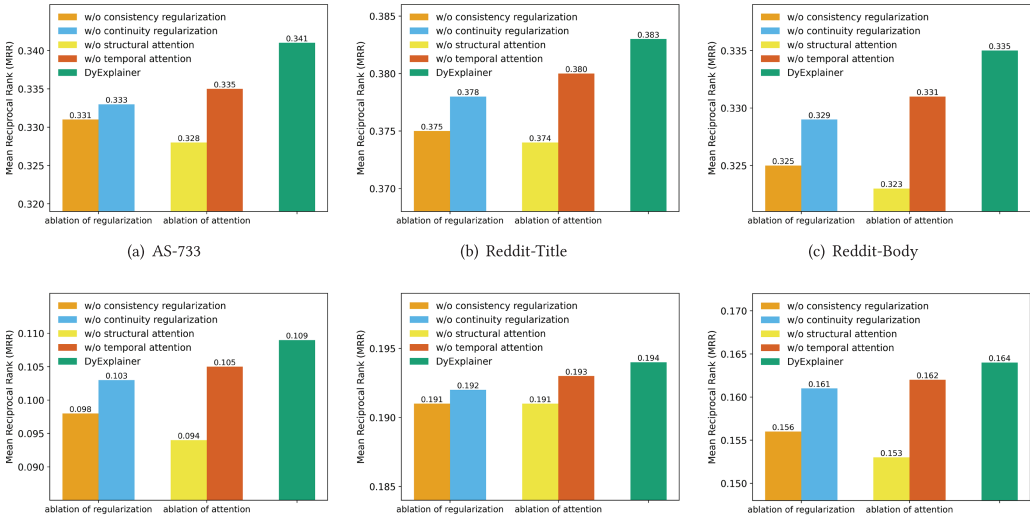
5.1.5 Computing Environment. We implemented all models using PyTorch [38], PyTorch Geometric [9], and Scikit-learn [39]. All datasets used in the experiments are obtained from [21]. We conduct the experiments on a server with four NVIDIA RTX A6000 GPUs (48 GB memory each).

5.2 Link Prediction Performance

Table 3 showcases the MRR results for all compared methods on all datasets, which were all trained through live updating. The results are obtained by averaging the MRR of the most recent 60% snapshots on the test datasets. The buffer size for DyExplainer was set to five across all datasets, and the attention module was fine-tuned for four epochs. The tradeoff parameter α is set to 0.5 for AS-733, Reddit-Body, and UCI-Message; 0.2 for Reddit-Title; 0.6 for Bitcoin-OTC; and 0.8 for Bitcoin-Alpha. The ROLAND method outperforms the other baselines on most datasets. DyExplainer surpasses the best baseline on four datasets, showing a 7.89% improvement on Bitcoin-Alpha, and performs similarly to the best baseline on the remaining datasets. This effectively demonstrates the Rashomon set theory [44, 60] and highlights the benefits of seeking a simple and understandable model, which can result in improved robustness and better overall performance.

5.3 Ablation Study

To present deep insights into the proposed method, we conducted multiple ablation studies on the six datasets to empirically verify the effectiveness of the proposed explainable aggregations



(a) AS-733 (b) Reddit-Title (c) Reddit-Body

(d) UCI-Message (e) Bitcoin-OTC (f) Bitcoin-Alpha

Fig. 4. Ablation studies on six datasets.

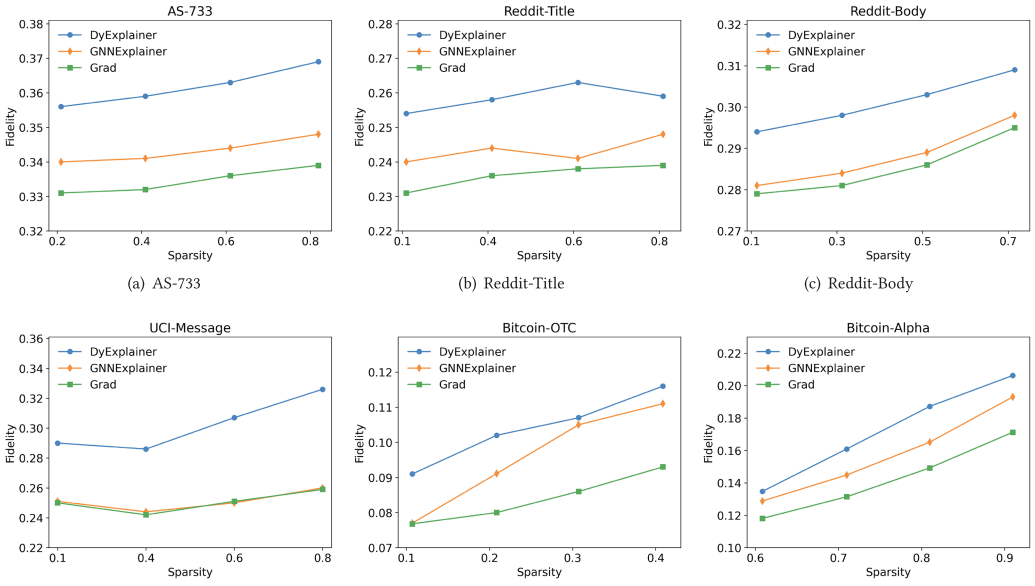
and the contrastive regularizations. Specifically, we compare DyExplainer with variants. (1) w/o structural attention and w/o temporal attention are the DyExplainer without one of the attention modules in the explainable aggregations. (2) w/o consistency regularization and w/o continuity regularization refers to DyExplainer without one of the contrastive regularization terms in the loss.

We show the comparison between DyExplainer and the ablation variants in Figure 4. From the figure, we observe that: (1) the performance of w/o structural attention is much worse than w/o temporal attention; (2) the performance of w/o consistency regularization is much worse than w/o continuity regularization. It is evident from both (1) and (2) that the topological information within a single snapshot holds greater sway over the prediction outcome, as compared to the temporal dependencies that exist between snapshots. (3) The results of the ablation studies performed on regularization and attention further reinforce the superiority of our approach over these alternatives.

5.4 Explanations Performance

In order to demonstrate the reliability of the explanations provided by DyExplainer, we conduct quantitative evaluations that compare our approach with various baselines across six datasets characterized by a limited number of edges: AS-733, Reddit-Title, Reddit-Body, UCI-Message, Bitcoin-OTC, and Bitcoin-Alpha. Specifically, we adopt the Fidelity vs. Sparsity metrics for our evaluation, in accordance with the methodology described in [42, 68]. The Fidelity metric assesses the accuracy with which the explanations reflect the significance of various factors to the model's predictions, while the Sparsity metric quantifies the proportion of structures that are deemed critical by the explanation techniques. For a fair comparison, for all three methods, we use a trained dynamic GNN [65] as the base model to calculate the predicted probability.

The baseline method, GNNExplainer, was not originally developed for dynamic graph settings. Nevertheless, for a fair comparison, we assess the Fidelity of both DyExplainer and GNNExplainer using only the graph at the final snapshot, despite the fact that DyExplainer provides explanations for all snapshots stored in a buffer. GNNExplainer identifies the most influential nodes within a



(a) AS-733 (b) Reddit-Title (c) Reddit-Body

(d) UCI-Message (e) Bitcoin-OTC (f) Bitcoin-Alpha

Fig. 5. The quantitative studies for different explanation methods on six datasets.

k-hop neighborhood and generates a mask to highlight these nodes for a given prediction node. As our DyExplainer provides a global attention mechanism for the entire graph, for a fair comparison, we calculate the average Fidelity of each mask produced by GNNExplainer with respect to all of the edges. To obtain the Fidelity and Sparsity metrics for each mask, we select a subset of the top-ranked edges based on the weights from the explanation mask and use this subset to create a new graph. The Fidelity of an edge is defined as the difference between the predicted probability of that edge on the new graph and the global graph. For DyExplainer, we directly select the top-ranked edges based on the attention weights to form a new graph, as there is only one global attention mechanism that provides the explanation.

The comparisons of Fidelity and Sparsity are presented in Figure 5. The evaluation of various methods is based on their Fidelity scores under comparable levels of Sparsity, as the Sparsity level cannot be precisely controlled. From the figure, we see that for all six datasets, the Fidelity of these methods increases as Sparsity increases. This is due to the calculation of Fidelity as the difference between the predicted probability of the model on the reduced graph and the original graph, leading to higher Fidelity values with greater Sparsity. Furthermore, GNNExplainer demonstrates superior Fidelity compared to Grad on AS-733, Reddit-Title, Bitcoin-OTC, and Bitcoin-Alpha, while performing similarly on Reddit-Body and UCI-Message. Additionally, DyExplainer outperforms both methods on all six datasets at varying Sparsity levels, revealing that it provides more accurate explanations.

5.5 Hyper-parameter Analysis

The tradeoff parameter α in Equation (11) controls the strength of consistency and continuity regularizations. We analyze the performance of DyExplainer with respect to α on datasets AS-733, Reddit-Title, and Reddit-Body in Figure 6. The figure shows that as α increases, the MRRs of DyExplainer initially improve but then decline as α continues to increase.

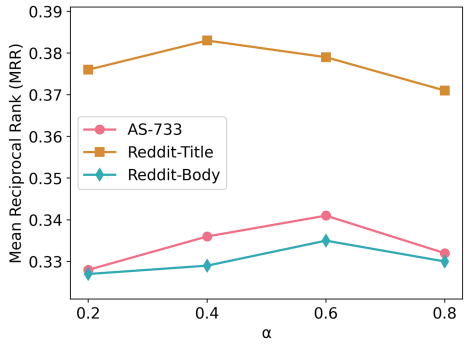


Fig. 6. Analysis of parameter α.

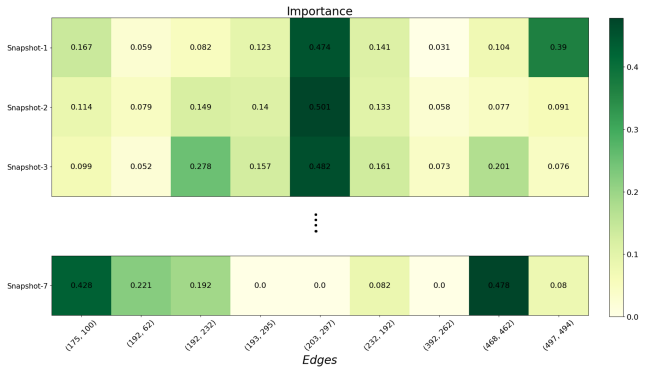


Fig. 7. Heat map of edge importance over time. We sample some edges shared by continuous snapshots and show their attention values. 0.0 means this edge does not appear in this snapshot.

5.6 Case Study I

To show the evolving and continuity of underlying patterns the DyExplainer detects from the dynamic graph, we visualize the attention values of some edges at different snapshots on the Bitcoin-Alpha dataset in Figure 7. From this figure, we observe that the edges $(192, 62)$, $(193, 295)$, $(203, 297)$, and $(232, 192)$ have close importance on snapshots 1–3. It indicates the temporal continuity of edge importance exists in dynamic graphs and our intuition is reasonable. The edges $(175, 100)$, $(192, 62)$, and $(468, 462)$ are less important at snapshots 1–3 while becoming more important at snapshot 7. It infers that the importance of an edge is evolving along with the timesteps.

5.7 Case Study II

We further visualize the temporal attention to show which historical snapshot has more influence on the current. The importance of snapshots provided by the temporal attention is shown in Figure 8. From the figure, we see that the most important snapshot to the current is snapshot 32.

To see how the patterns have an effect, we randomly pick an edge $(623, 26)$ at snapshot 35 to investigate the local structure of this edge at snapshot 32. The visualization of nodes 623 and 26 is shown in Figure 9. Note that these two nodes are unconnected at snapshot 32. The widths of edges are according to the weights of structural attention. Thicker ones mean large weights. From the figure, we can find that there are three clusters of substructures with larger weights, which can be viewed as important patterns that DyExplainer detects.

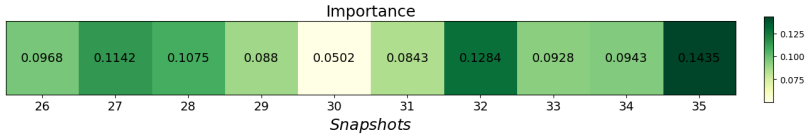


Fig. 8. Heat map of temporal importance over time. The current snapshot is at 35. The buffer size is set to 10. It means we consider the importance of the previous snapshot from 27 to 34.

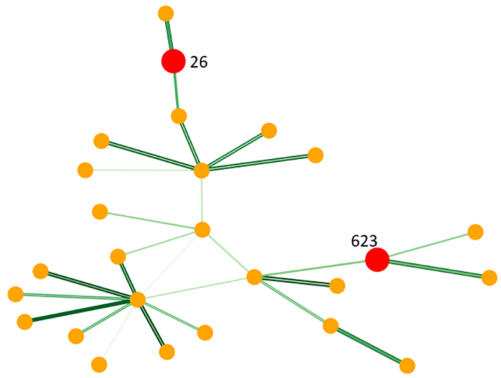


Fig. 9. Local structure of nodes 26 and 623 on snapshot 32. The red points are the two nodes we study, and the orange points are their 3-hop neighborhoods. The green edges are obtained from the structural attention weights at snapshot 32.

6 Discussion

In this work, we tackle the challenge of developing explainable dynamic GNNs, with our approach built upon the foundational architecture of GNNs. Recently, the rapid advancements in **Large Language Models (LLMs)** have led to impressive success in processing natural language across various domains. In this era of LLMs, numerous efforts have been made to leverage these models to enhance GNN-based architectures for downstream tasks. For example, some modern studies have investigated the integration of GNNs with LLMs to more effectively capture and enrich the textual attributes embedded within graphs [2, 4, 26, 50, 62, 72]. Alternative approaches have sought to represent graphs linguistically for direct processing by LLMs, such as [5, 10, 13, 55]. Recent mainstream works also explored aligning graphs as node token sequences with natural language token sequences when inputting them to LLMs. These approaches discard the traditional graph encoder and adopt a specific arrangement of graph token sequences, along with carefully designed embeddings of graph tokens in prompts. InstructGLM [62] combines natural language instructions with graph embeddings to fine-tune LLMs. GraphGPT [49] integrates text descriptions with a self-supervised graph transformer to incorporate graph data into LLMs. LLaGA [3] adopts node-level templates to restructure graph data into organized sequences, which are then mapped into the token embedding space. This approach helps guide LLMs in processing graph-structured data with improved explainability. While LLM-enhanced graph learning has shown promising results across various downstream graph-related tasks and provided valuable insights for graph explainability, most of these efforts focus on static graphs. Addressing the challenges of explaining dynamic graphs with evolving structures using LLMs necessitates more efficient interactions between graph token sequences and LLM token sequences, a direction we leave for future work.

7 Conclusion

GNNs have become a popular research topic due to their ability to capture representations from graph-structured data. However, the intrinsic black-box nature of these models poses a considerable challenge, hindering their practical use in crucial scenarios where trust and transparency are of utmost importance. Despite the commendable efforts aimed at explaining GNNs, much of the existing work is limited to static graphs, leaving the explanation of dynamic GNNs largely underexplored. To address this issue, we present DyExplainer, a pioneering approach to explaining dynamic GNNs in real-time. DyExplainer leverages a dynamic GNN backbone to extract representations at each snapshot, concurrently exploring structural relationships and temporal dependencies through a sparse attention mechanism. To ensure structural consistency and temporal continuity in the explanation, our approach incorporates contrastive learning techniques and a buffer-based live-updating scheme. The results of our experiments showcase the superiority of DyExplainer, providing a faithful explanation of the model predictions while concurrently improving the accuracy of the model, as evidenced by the link prediction task.

References

[1] Federico Baldassarre and Hossein Azizpour. 2019. Explainability techniques for graph convolutional networks. arXiv:1905.13686. Retrieved from https://arxiv.org/abs/1905.13686

[2] Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. 2023. Graphllm: Boosting graph reasoning ability of large language model. arXiv:2310.05845. Retrieved from https://arxiv.org/abs/2310.05845

[3] Runjin Chen, Tong Zhao, Ajay Jaiswal, Neil Shah, and Zhangyang Wang. 2024. Llaga: Large language and graph assistant. arXiv:2402.08170. Retrieved from https://arxiv.org/abs/2402.08170

[4] Zhikai Chen, Haitao Mao, Hongzhi Wen, Haoyu Han, Wei Jin, Haiyang Zhang, Hui Liu, and Jiliang Tang. 2023. Label-free node classification on graphs with large language models (LLMS). arXiv:2310.04668. Retrieved from https://arxiv.org/abs/2310.04668

[5] Sitao Cheng, Ziyuan Zhuang, Yong Xu, Fangkai Yang, Chaoyun Zhang, Xiaoting Qin, Xiang Huang, Ling Chen, Qingwei Lin, Dongmei Zhang, et al. 2024. Call me when necessary: LLMs can efficiently and faithfully reason over structured environments. arXiv:2403.08593. Retrieved from https://arxiv.org/abs/2403.08593

[6] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. arXiv:1409.1259. Retrieved from https://arxiv.org/abs/1409.1259

[7] Enyan Dai and Suhang Wang. 2021. Towards self-explainable graph neural network. In *CIKM*, 302–311.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, Vol. 29.

[9] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *RLGM@ICLR*.

[10] Yu Gu, Yiheng Shu, Hao Yu, Xiao Liu, Yuxiao Dong, Jie Tang, Jayanth Srinivasa, Hugo Latapie, and Yu Su. 2024. Middleware for LLMS: Tools are instrumental for language agents in complex environments. arXiv:2402.14672. Retrieved from https://arxiv.org/abs/2402.14672

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*, Vol. 30.

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

[13] Jin Huang, Xingjian Zhang, Qiaozhu Mei, and Jiaqi Ma. 2023. Can LLMS effectively leverage graph structural information: when and why. arXiv:2309.16595. Retrieved from https://arxiv.org/abs/2309.16595

[14] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. 2022. Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering* 35, 7 (2022), 6968–6972.

[15] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. arXiv:1611.01144. Retrieved from https://arxiv.org/abs/1611.01144

[16] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. arXiv:1609.02907. Retrieved from https://arxiv.org/abs/1609.02907

[17] Srijan Kumar, William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2018. Community interaction and conflict on the web. In *WWW*, 933–943.

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

[19] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *ICDM*. IEEE, 221–230.
[20] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *ACM SIGKDD*, 177–187.
[21] Jure Leskovec and Andrej Krevl. 2014. SNAP datasets: Stanford large network dataset collection. Retrieved from http://snap.stanford.edu/data
[22] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. arXiv:1511.05493. Retrieved from https://arxiv.org/abs/1511.05493
[23] Ke Liang, Lingyuan Meng, Meng Liu, Yue Liu, Wenxuan Tu, Siwei Wang, Sihang Zhou, Xin wang Liu, and Fuchun Sun. 2022. Reasoning over different types of knowledge graphs: Static, temporal and multi-modal. arXiv:2212.05767. Retrieved from https://arxiv.org/abs/2212.05767
[24] Meng Liu, Yue Liu, Ke Liang, Siwei Wang, Sihang Zhou, and Xinwang Liu. 2023. Deep temporal graph clustering. arXiv:2305.10738. Retrieved from https://arxiv.org/abs/2305.10738
[25] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and choose: A GNN-based imbalanced learning approach for fraud detection. In *WWW*, 3168–3177.
[26] Zhiyuan Liu, Sihang Li, Yanchen Luo, Hao Fei, Yixin Cao, Kenji Kawaguchi, Xiang Wang, and Tat-Seng Chua. 2023. MolCA: Molecular graph-language modeling with cross-modal projector and uni-modal adapter. arXiv:2310.12798. Retrieved from https://arxiv.org/abs/2310.12798
[27] Zheng Liu, Xiaohan Li, Hao Peng, Lifang He, and S. Yu Philip. 2020. Heterogeneous similarity graph neural network on electronic health records. In *Big Data*. IEEE, 1196–1205.
[28] Christos Louizos, Max Welling, and Diederik P. Kingma. 2017. Learning sparse neural networks through L_0 regularization. arXiv:1712.01312. Retrieved from https://arxiv.org/abs/1712.01312
[29] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized explainer for graph neural network. In *NIPS*, Vol. 33, 19620–19631.
[30] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *ACM SIGKDD*, 723–731.
[31] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. arXiv:1611.00712. Retrieved from https://arxiv.org/abs/1611.00712
[32] Franco Manessi, Alessandro Rozza, and Mario Manzo. 2020. Dynamic graph convolutional networks. *Pattern Recognition* 97 (2020), 107000.
[33] Miller McPherson, Lynn Smith-Lovin, and James M. Cook. 2001. Birds of a feather: Homophily in social networks. *Annual Review of Sociology* 27, 1 (2001), 415–444.
[34] Siqi Miao, Mia Liu, and Pan Li. 2022. Interpretable and generalizable graph learning via stochastic attention mechanism. In *ICML*. PMLR, 15524–15543.
[35] Meike Nauta, Jan Trienes, Shreyasi Pathak, Elisa Nguyen, Michelle Peters, Yasmin Schmitt, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. 2022. From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable AI. arXiv:2201.08164. Retrieved from https://arxiv.org/abs/2201.08164
[36] Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology* 60, 5 (2009), 911–932.
[37] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*, Vol. 34, 5363–5370.
[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NIPS*, Vol. 32.
[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[40] Hao Peng, Hongfei Wang, Bowen Du, Md Zakirul Alam Bhuiyan, Hongyuan Ma, Jianwei Liu, Lihong Wang, Zeyu Yang, Linfeng Du, Senzhang Wang, et al. 2020. Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting. *Information Sciences* 521 (2020), 277–290.
[41] Jingyu Peng, Qi Liu, Linan Yue, Zaixi Zhang, Kai Zhang, and Yunhao Sha. 2024. Towards few-shot self-explaining graph neural networks. In *ECML PKDD*. Springer, 109–126.
[42] Phillip E. Pope, Soheil Kolouri, Mohammad Rostami, Charles E. Martin, and Heiko Hoffmann. 2019. Explainability methods for graph convolutional neural networks. In *CVPR*, 10772–10781.

[43] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. arXiv:2006.10637. Retrieved from https://arxiv.org/abs/2006.10637
[44] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. 2022. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys* 16 (2022), 1–85.
[45] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM*, 519–527.
[46] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. 2020. Interpreting graph neural networks for NLP with differentiable edge masking. arXiv:2010.00577. Retrieved from https://arxiv.org/abs/2010.00577
[47] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *ICONIP*. Springer, 362–373.
[48] Yongduo Sui, Xiang Wang, Jiancan Wu, Min Lin, Xiangnan He, and Tat-Seng Chua. 2022. Causal attention for interpretable and generalizable graph classification. In *SIGKDD*, 1696–1705.
[49] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2024. Graphgpt: Graph instruction tuning for large language models. In *ACM SIGIR*, 491–500.
[50] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Long Xia, Dawei Yin, and Chao Huang. 2024. Higpt: Heterogeneous graph language model. In *ACM SIGKDD*.
[51] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. 2021. Unsupervised representation learning for time series with temporal neighborhood coding. In *ICLR*.
[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, Vol. 30.
[53] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. arXiv:1710.10903. Retrieved from https://arxiv.org/abs/1710.10903
[54] Minh Vu and My T. Thai. 2020. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. In *NeurIPS*, Vol. 33, 12225–12235.
[55] Qinyong Wang, Zhenxiang Gao, and Rong Xu. 2023. Graph agent: Explicit reasoning agent for graphs. arXiv:2310.16421. Retrieved from https://arxiv.org/abs/2310.16421
[56] Tianchun Wang, Farzaneh Mirzazadeh, Xiang Zhang, and Jie Chen. 2023. Gc-flow: A graph-based flow network for effective clustering. In *ICML*. PMLR, 36157–36173.
[57] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. 2020. Traffic flow prediction via spatial temporal graph neural network. In *WWW*, 1082–1092.
[58] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph neural networks in recommender systems: A survey. *ACM Computing Surveys* 55, 5 (2022), 1–37.
[59] Jiaxuan Xie, Yezi Liu, and Yanning Shen. 2022. Explaining dynamic graph neural networks via relevance back-propagation. arXiv:2207.11175. Retrieved from https://arxiv.org/abs/2207.11175
[60] Rui Xin, Chudi Zhong, Zhi Chen, Takuya Takagi, Margo Seltzer, and Cynthia Rudin. 2022. Exploring the whole rashomon set of sparse decision trees. In *NeurIPS*.
[61] Sibei Yang, Guanbin Li, and Yizhou Yu. 2019. Dynamic graph attention for referring expression comprehension. In *ICCV*, 4644–4653.
[62] Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. 2023. Natural language is all a graph needs. arXiv:2308.07134. Retrieved from https://arxiv.org/abs/2308.07134
[63] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *NeurIPS*, 9240–9251.
[64] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NIPS*, Vol. 31.
[65] Jiaxuan You, Tianyu Du, and Jure Leskovec. 2022. ROLAND: Graph learning framework for dynamic graphs. In *ACM SIGKDD*, 2358–2366.
[66] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv:1709.04875. Retrieved from https://arxiv.org/abs/1709.04875
[67] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2022. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 5 (2022), 5782–5799.
[68] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On explainability of graph neural networks via subgraph explorations. In *ICML*. PMLR, 12241–12252.
[69] Chun-Yang Zhang, Zhi-Liang Yao, Hong-Yu Yao, Feng Huang, and C. L. Philip Chen. 2022. Dynamic representation learning via recurrent graph neural networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 53, 2 (2022), 1284–1297.
[70] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *NeurIPS*, Vol. 31.

[71] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. 2022. Protgnn: Towards self-explaining graph neural networks. In *AAAI*, Vol. 36, 9127–9135.
[72] Haiteng Zhao, Shengchao Liu, Ma Chang, Hannan Xu, Jie Fu, Zhihong Deng, Lingpeng Kong, and Qi Liu. 2023. Gimlet: A unified graph-text model for instruction-based molecule zero-shot learning. In *NeurIPS*.
[73] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2019. T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems* 21, 9 (2019), 3848–3858.
[74] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. 2019. AddGraph: Anomaly detection in dynamic graph using attention-based temporal GCN. In *IJCAI*, Vol. 3, 7.

Received 23 February 2024; revised 7 November 2024; accepted 8 March 2025