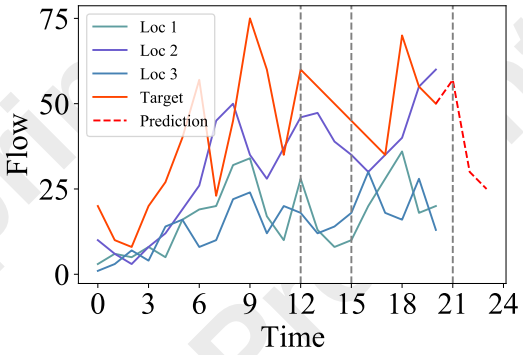
# `DGExplainer`: Explaining Dynamic Graph Neural Networks via Relevance Back-propagation*

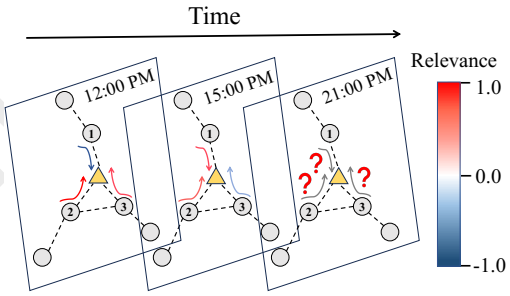
**Yezi Liu** , **Jiaxuan Xie** , **Yanning Shen**†
University of California, Irvine
{yezil3, yannings}@uci.edu, Knxie@outlook.com

## Abstract

Dynamic graph neural networks (dynamic GNNs) are highly effective for analyzing time-varying graph-structured data. However, their black-box nature often makes it difficult for users to understand the reasoning behind their predictions, which can limit their applications. Although recent years have seen increased research on explaining GNNs, most existing studies focus on static graphs. Explaining dynamic GNNs is uniquely challenging due to their spatial and temporal structures. Directly applying methods designed for static graphs to dynamic graphs is not feasible, since these methods overlook temporal dependencies. To address this gap, we propose `DGExplainer`, a novel approach that provides reliable explanations for dynamic GNN predictions. `DGExplainer` uses relevance back-propagation across both time-wise and layer-wise. First, it captures temporal information by tracking the relevance of node representations backward through time. Then, at each step, layer-wise relevance within the graph module is calculated by redistributing node representation relevance along the back-propagation path. Quantitative and qualitative experiments on six real-world datasets demonstrate that `DGExplainer` effectively identifies critical nodes for link prediction and node regression tasks in dynamic GNNs.




(a) Traffic flow prediction.




(b) Explaining traffic-flow prediction of dynamic GNNs.

Figure 1: The diagram of the explanation task of dynamic GNNs on traffic flow data.

## 1 Introduction

Dynamic GNNs have achieved significant success in practical applications such as social network analysis [Zhu *et al.*, 2016], transportation forecasting [Gui *et al.*, 2020], and pandemic forecasting [Kapoor *et al.*, 2020]. However, since most of the dynamic GNNs [Ma *et al.*, 2020; Li *et al.*, 2017] are developed without interpretability, they are treated as black-boxes. Without understanding the underlying mechanisms behind their predictions, dynamic GNNs cannot be fully trusted, preventing their use in critical applications. In order to safely and trustworthily employ dynamic GNN models, it is important to provide both accurate predictions and human-understandable explanations.

Recent studies have extensively explored explanation techniques for static GNNs. These techniques include approximation-based methods [Baldassarre and Azizpour, 2019; Pope *et al.*, 2019], which use gradients or surrogate functions to approximate the output of a local model. Perturbation-based approaches [Ying *et al.*, 2019; Luo *et al.*, 2020] explain static GNNs by masking specific features to observe their impact on the model's output. Gradient-based

*Appendix available at https://github.com/yezil3/DGExplainer_IJCAI/blob/main/IJCAI_appendix.pdf

†Corresponding author.

methods [Selvaraju *et al.*, 2017] adopt the additive assumption of feature values or gradients to measure the importance of input features. Further relevant research on explaining static GNNs can be found in Appendix A.13. However, these methods overlook the critical temporal dimension crucial for explaining dynamic GNNs. Applying static-graph explanation frameworks to dynamic graphs yields discrete, snapshot-based explanations that ignore temporal context.

Explaining dynamic GNNs can be challenging. We illustrate this process in Figure 1. The prediction task, shown in Figure 1a, aims to forecast future traffic flows (denoted by dashed lines) at different locations based on historical observations (denoted by solid lines). This spatial-temporal data is modeled as a dynamic graph, represented in Figure 1b, where each graph snapshot records traffic flows at different time steps (e.g., 12:00 PM, 3:00 PM, and 9:00 PM). In each graph snapshot, a dashed line between two nodes indicates a commute between locations, and an arrow represents traffic flows, contributing to the prediction for the target location (denoted by a yellow triangle). The explanation task aims to determine the influence of other locations on the prediction of the target location. The polarity of the influence is denoted by the color of the arrows: blue indicates a positive correlation, while red indicates a negative correlation, with the darkness of the color indicating the strength of the influence. Dynamic graph data involves both temporal and spatial patterns, which dynamic GNNs capture using separate temporal and spatial modules. This makes explanation challenging, as it requires tracing how the input influences the output through these complex components.

To address this challenge, we propose `DGExplainer` (Dynamic Graph Neural Network Explainer). `DGExplainer` uses layer-wise relevance propagation (LRP), which was originally introduced for image classifiers [Bach *et al.*, 2015]. LRP can assign relevance scores without the need for a surrogate model or additional optimization. Unlike methods that only consider individual nodes or edges, LRP evaluates sequences of edges or walks, making it especially well-suited for dynamic GNNs. Our framework operates in three main steps. First, `DGExplainer` decomposes the prediction of a dynamic GNN to determine how time-related modules contribute to that prediction, using relevance back-propagation. Second, it propagates relevance scores through the graph-related modules (e.g., a GCN) layer by layer at each time step to calculate the importance of input features. Finally, `DGExplainer` aggregates the relevance scores from both steps to produce the final relevance of the node features. The contributions of our work are as follows:

- The proposed method explains the predictions of dynamic graph neural networks, marking one of the pioneering efforts to tackle this challenge.
- This paper is the first to derive LRP for time-varying modules. These hand-derived derivations are essential as they serve as a reference for adjusting or writing code when the provided functions fail, and adaptations are needed.
- Experimental results across six datasets and three quantitative metrics show that `DGExplainer` provides faithful explanations. Furthermore, qualitative analysis indicates that `DGExplainer` outperforms other baseline methods in effectively explaining dynamic GNNs.

## 2 Problem Definition

This paper solves the problem of explaining dynamic GNNs by computing the relevances of input features. The proposed method first redistributes the prediction from the last layer to the relevance of hidden representations. Then, we use LRP to back-propagate the relevances through *time-related* and *graph-related* modules, finally reaching the input layer to obtain the relevances of input features of each node.

We study a series of input graphs $\mathcal{G} = \{\mathbf{X}_t, \mathbf{A}_t\}_{t-1}^{T}$, where $T$ is the length of the sequence. Each graph at time $t$, $\mathcal{G}_t = \{\mathbf{X}_t, \mathbf{A}_t\}$, consists of a feature matrix $\mathbf{X}_t \in \mathbb{R}^{N \times D}$ and an adjacency matrix $\mathbf{A}_t \in \mathbb{R}^{N \times N}$. Here, $N = |\mathcal{V}_t|$ denotes the number of nodes, and $D$ is the feature dimension. The feature vector for node $i$ at time $t$ is $\mathbf{x}_t^i = (\mathbf{X}_t^{(i,:)})^\top \in \mathbb{R}^D$, which corresponds to the $i$-th row of $\mathbf{X}_t$. Without loss of generality, $\mathbf{A}^{(i,j)}$ denotes the entry at the $i$-th row and $j$-th column of the adjacency matrix $\mathbf{A}$, and $\mathbf{x}^{(i)}$ denotes the $i$-th entry of the vector $\mathbf{x}$. The relevance of an element $k$, which can be a node, an edge, a feature, etc., is represented by $R_k$. Additionally, $R_{k_1 \leftarrow k_2}$ denotes the relevance of $k_1$ distributed from $k_2$. The *problem* of explaining dynamic GNNs involves identifying the subgraph within $\mathcal{G}$, which consists of nodes and edges that are most important at a specific time step $t$, given a dynamic GNN model $f(\mathcal{G})$.

## 3 Preliminaries

In this section, we present the preliminaries relevant to our proposed method. We begin with an overview of dynamic GNNs in Section 3.1, followed by an introduction to layer-wise relevance propagation in Section 3.3.

### 3.1 Dynamic Graph Neural Networks

Dynamic GNNs [Skarding *et al.*, 2021; Zhang *et al.*, 2022] take a sequence of graphs as input and output representations of topology, nodes, and/or edges. A notable approach involves co-training a GNN with a recurrent neural network (RNN), referred to as a GNN-RNN model. Examples include GCN-GRU [Zhao *et al.*, 2019], ChebNet-LSTM [Seo *et al.*, 2018], and GCN-RNN [Pareja *et al.*, 2020]. Detailed related work about dynamic GNNs can be found in Appendix A.13. Despite the introduction of various methods, recent approaches still do not consistently outperform the GCN-GRU model [Pareja *et al.*, 2020]. Therefore, in this work, we choose to use the GCN-GRU model as the basis for elaborating our method. In addition to explaining the GCN-GRU model, we also apply `DGExplainer` to other dynamic GNNs that utilize different GNN or RNN architectures. Detailed results of these experiments can be found in Appendix A.7.

### 3.2 The GCN-GRU Model–Forward Pass

In the GCN-GRU model, the GCN module first encodes input features of the current time step, capturing dependencies between nodes. These encoded features are then passed to the

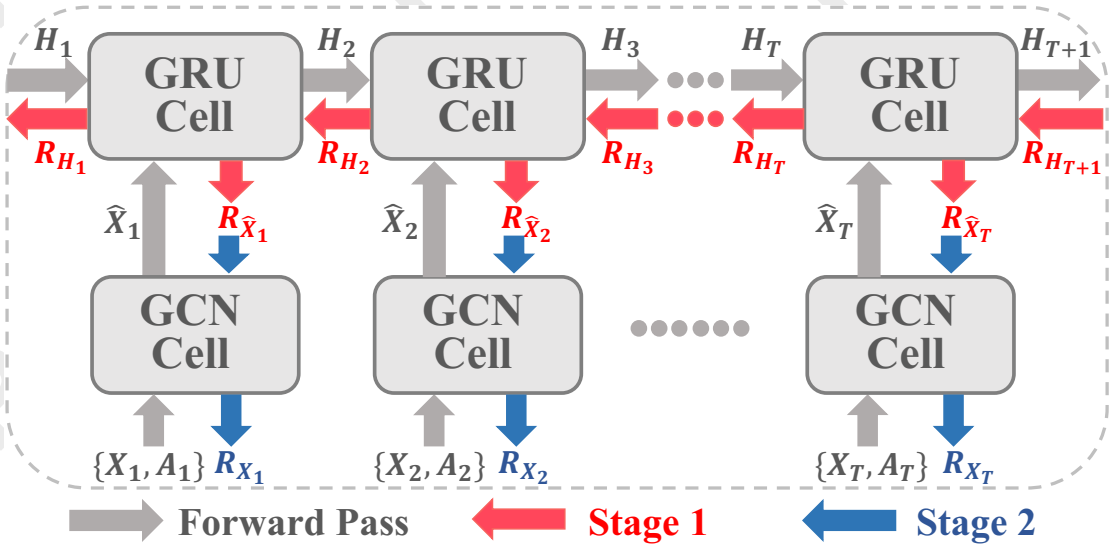



Figure 2: The network structure of the GCN-GRU model and the back-propagation of the relevances. Note that the GRU cells and GCN cells share the same parameters. $\{\mathbf{H}_t\}_{t=1}^{T+1}$, $\{\mathbf{X}_t\}_{t=1}^{T}$, $\{\hat{\mathbf{X}}_t\}_{t=1}^{T}$, $\{\mathbf{A}_t\}_{t=1}^{T}$ represent the hidden states, input features, GCN-encoded features, and adjacency matrices at different time steps, respectively.

GRU module, which captures temporal dependencies across different time steps. Below, we outline the forward process of the GCN-GRU.

**(a) The Graph Convolutional Network (GCN) module:** GCNs represent a node using local information from its surrounding neighbors [Kipf and Welling, 2016]. This graph convolution process is formulated as follows:

$$\mathbf{F}_t^{(l+1)} = \sigma(\mathbf{V}_t \mathbf{F}_t^{(l)} \mathbf{W}_t^{(l)}). \tag{1}$$

Here, $\mathbf{V}_t := \tilde{\mathbf{D}}_t^{-\frac{1}{2}} \tilde{\mathbf{A}}_t \tilde{\mathbf{D}}_t^{-\frac{1}{2}}$ is the normalized adjacency matrix, where $\tilde{\mathbf{A}}_t = \mathbf{A}_t + \mathbf{I}_N$ and $\tilde{\mathbf{D}}_t = \mathbf{D}_t + \mathbf{I}_N$. The matrix $\mathbf{D}_t$ is the degree matrix, defined as $\mathbf{D}_t^{(i,i)} = \sum_j \mathbf{A}_t^{(i,j)}$, and $\mathbf{I}_N$ is an identity matrix of size $N$. The output at the $l$-th layer is denoted as $\mathbf{F}_t^{(l)}$, with the initial layer output $\mathbf{F}_t^{(0)} = \mathbf{X}_t$. Assuming the GCN has $L$ layers, the final node representation at time step $t$, which contains the graph structural information, is denoted as $\hat{\mathbf{X}}_t = \mathbf{F}_t^{(L)}$. The GCN-encoded features from all time steps $\{\hat{\mathbf{X}}_t\}_{t=1}^{T}$ are then fed into a GRU.

**(b) The Gated Recurrent Unit (GRU) module:** GRU is a variant of the RNN designed to learn long-term dependencies using two selective gates [Cho *et al.*, 2014]. In the GRU, each cell processes an input $\hat{\mathbf{x}}_t = (\hat{\mathbf{X}}_t^{(i,:)})^\top$ and a hidden state $\mathbf{h}_t = (\mathbf{H}_t^{(i,:)})^\top$. The update rule for a GRU cell is as follows:

$$\mathbf{r} = \sigma\left(\mathbf{W}_{ir}\hat{\mathbf{x}}_t + \mathbf{b}_{ir} + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_{hr}\right), \tag{2a}$$

$$\mathbf{z} = \sigma\left(\mathbf{W}_{iz}\hat{\mathbf{x}}_t + \mathbf{b}_{iz} + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_{hz}\right), \tag{2b}$$

$$\mathbf{n} = \tanh\left(\mathbf{W}_{in}\hat{\mathbf{x}}_t + \mathbf{b}_{in} + \mathbf{r} \odot (\mathbf{W}_{hn}\mathbf{h}_{t-1} + \mathbf{b}_{hn})\right), \tag{2c}$$

$$\mathbf{h}_t = (1 - \mathbf{z}) \odot \mathbf{h}_{t-1} + \mathbf{z} \odot \mathbf{n}, \tag{2d}$$

where $\mathbf{W}_{ir}$, $\mathbf{W}_{hr}$, $\mathbf{W}_{hz}$, $\mathbf{W}_{in}$, $\mathbf{W}_{hn}$, $\mathbf{b}_{ir}$, $\mathbf{b}_{hr}$, $\mathbf{b}_{hz}$, $\mathbf{b}_{in}$, and $\mathbf{b}_{hn}$ are learnable parameters in GRU, $\sigma(\cdot)$ is an activation function, and $\odot$ is an element-wise product operation.

### 3.3 Layer-wise Relevance Propagation

In this paper, we consider neural networks consisting of layers of neurons. The output $x_{k_2}$ of a neuron $k_2$ is a non-linear activation function $g$ as given by:

$$x_{k_2} = g\left(\sum_{k_1} w_{k_1 k_2} x_{k_1} + b\right). \tag{3}$$

Assume that we know the relevance $R_{k_2}^{(l+1)}$ of a neuron $k_2$ at network layer $(l+1)$ for the classification decision $f(x)$, then we like to decompose this relevance into messages $R_{k_1 \leftarrow k_2}^{(l,l+1)}$ sent to those neurons $k_1$ at the layer $l$ which provide inputs to neuron $k_2$ such that eq. (4) holds.

$$R_{k_2}^{(l+1)} = \sum_{k_1 \in (l)} R_{k_1 \leftarrow k_2}^{(l,l+1)}. \tag{4}$$

We can then define the relevance of a neuron $k_1$ at layer $l$ by summing all messages from neurons at layer $(l+1)$ as in eq. (5):

$$R_{k_1}^{(l)} = \sum_{k_2 \in (l+1)} R_{k_1 \leftarrow k_2}^{(l,l+1)}. \tag{5}$$

The relevance of the output neuron at layer $M$ is $R_1^{(M)} = f(x)$. The pixel-wise scores are the resulting relevances of the input neurons $R_d^{(1)}$.

## 4 The Proposed `DGExplainer`

This section introduces the proposed `DGExplainer` framework (summarized in Figure 2), which explains dynamic GNN predictions by back-propagating relevance through both *time-varying* and *message-passing* reverse paths. By accounting for the structural and temporal information in dynamic graphs, `DGExplainer` computes the relevance of

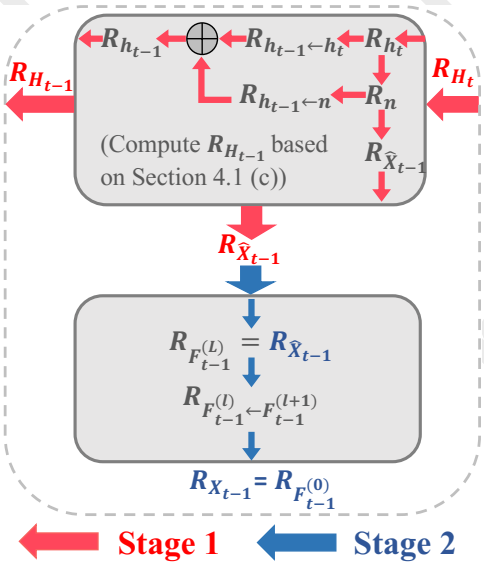



Figure 3: An illustration of DGExplainer computing feature relevance in a backward manner: the prediction is first back-propagated through the GRU, followed by the GCN.

each input feature, offering deeper insights into how the model arrives at its predictions.

DGExplainer operates in two stages: **Stage 1:** *Compute Relevance in the GRU Module.* In this stage, relevance scores are back-propagated through the time-related module (GRU) along its reversed time paths. As a result, we obtain the relevance scores for the GCN-encoded features at every time step. **Stage 2:** *Back-Propagate Relevance in the GCN Module.* Next, the relevance scores from Stage 1 are taken as input and back-propagated through the graph-related module (e.g., GCN) along the inverse of its message-passing path. This allows us to determine the relevance of the original input features across all time steps, providing an explanation for the dynamic graphs.

### 4.1 Stage 1: Relevance Back-Propagation in GRU

**Stage 1** focuses on deriving the relevance of each GRU cell's inputs by propagating the output relevance backward in time. Specifically, DGExplainer first obtains the relevance of the output from the final GRU cell. Then, at each time step $t$, it uses the current cell's output relevance to compute the relevances of the two inputs: (1) the GCN-encoded feature and (2) the hidden state. The detailed process is illustrated below.

Given the final hidden state for a node, $R_{\mathbf{h}_T}$, where $\mathbf{h}_T = (\mathbf{H}_T^{(i,:)})^\top$, the objective is to compute the relevances of the inputs, $R_{\mathbf{h}_{t-1}}$ and $R_{\hat{\mathbf{x}}_{t-1}}$, from the relevance of the output, $R_{\mathbf{h}_t}$, for each GRU cell at time $t$. As described in Section 3.3, relevance back-propagation redistributes the activation of a descendant neuron to its predecessor neurons, with the relevance being proportional to the weighted activation value. Based on the dependencies among different components in the final step of the GRU, as shown in Equation (2d), we derive the relevance back-propagation for this step as follows:

$$R_{\mathbf{h}_{t-1}} = R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t} + R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}} + R_{\mathbf{h}_{t-1}\leftarrow\mathbf{z}} + R_{\mathbf{h}_{t-1}\leftarrow\mathbf{r}}. \quad (6)$$

Note that neurons $\mathbf{r}$ and $\mathbf{z}$ only receive messages from neuron $\mathbf{h}_{t-1}$, as shown in Equations (2a) and (2b). Consequently, their contribution to $\mathbf{h}_t$ can be merged into the contribution from $\mathbf{h}_{t-1}$, and their relevances can be regarded as constants. Notice that $\mathbf{h}_{t-1}$ is used to compute both $\mathbf{n}$ in Equation (2c) and $\mathbf{h}_t$ in Equation (2d). This reveals that the relevance $R_{\mathbf{h}_{t-1}}$ has two sources: $\mathbf{n}$ and $\mathbf{h}_t$. Based on the contributions from $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}$ and $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t}$, we can define $R_{\mathbf{h}_t}$ as follows:

$$R_{\mathbf{h}_t} = R_{\mathbf{h}_{t-1}} + R_{\mathbf{n}}, \quad (7)$$

Given that the relevance of a neuron is proportional to its activation at the same layer, i.e., $R_{k\leftarrow k_1} : R_{k\leftarrow k_2} = a_{k_1}^{(l)} : a_{k_2}^{(l)}$, we can derive the following based on Equation (2d):

$$\frac{R_{\mathbf{h}_{t-1}}}{R_{\mathbf{n}}} = \frac{a_{\mathbf{h}_{t-1}}}{a_{\mathbf{n}}} = \frac{\mathbf{z}\odot\mathbf{n}}{(1-\mathbf{z})\odot\mathbf{h}_{t-1}}. \quad (8)$$

We can conclude that if we derive $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t}$ and $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}$, we can then obtain $R_{\mathbf{h}_t}$. Therefore, we break down this problem into three steps: computing $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t}$, $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}$, and $R_{\mathbf{h}_{t-1}}$, as formulated below: **(a) Compute $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t}$:** Solving for Equations (7) and (8) obtains:

$$R_{\mathbf{h}_t\leftarrow\mathbf{n}} = \frac{\mathbf{z}\odot\mathbf{n}}{\mathbf{h}_t+\epsilon}\odot R_{\mathbf{h}_t}, \quad (9)$$

$$R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t} = \frac{(1-\mathbf{z})\odot\mathbf{h}_{t-1}}{\mathbf{h}_t+\epsilon}\odot R_{\mathbf{h}_t}, \quad (10)$$

where $\epsilon > 0$ is a constant introduced to keep the denominator non-zero. Notice that the only ancestor neuron of $\mathbf{n}$ is $\mathbf{h}_t$, so here $R_{\mathbf{n}\leftarrow\mathbf{h}_t}$ is actually $R_{\mathbf{n}}$, so in the following left of section, we use $R_{\mathbf{n}}$ for simplicity.

**(b) Compute $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}$:** From Equation (2c) we obtain:

$$\mathbf{n}_1 := \mathbf{W}_{in}\hat{\mathbf{x}}_t, \quad (11a)$$

$$\mathbf{n}_2 := \mathbf{r}\odot(\mathbf{W}_{hn}\mathbf{h}_{t-1}) = \mathbf{W}_{rn}\mathbf{h}_{t-1}, \quad (11b)$$

$$\mathbf{b}_{\mathbf{n}} := \mathbf{b}_{in} + \mathbf{r}\odot\mathbf{b}_{hn}. \quad (11c)$$

Then, their relevance satisfies:

$$R_{\mathbf{n}} = R_{\mathbf{n}_1} + R_{\mathbf{n}_2} + R_{\mathbf{b}_n}, \quad (12)$$

$$R_{\mathbf{n}_1} : R_{\mathbf{n}_2} : R_{\mathbf{b}_n} = \mathbf{n}_1 : \mathbf{n}_2 : \mathbf{b}_n. \quad (13)$$

Hence, $R_{\mathbf{n}_1}$ and $R_{\mathbf{n}_2}$ can be obtained.

$$R_{\hat{\mathbf{x}}_t\leftarrow\mathbf{n}_1} = \sum_k \frac{\mathbf{W}_{in}^{(k,j)}\hat{\mathbf{x}}_t^{(j)}}{\epsilon+\sum_i \mathbf{W}_{in}^{(k,i)}\hat{\mathbf{x}}_t^{(i)}} R_{\mathbf{n}_1}^{(k)}. \quad (14)$$

Since $\mathbf{h}_{t-1}$ influences only $\mathbf{n}_2$ among the three components of $\mathbf{n}$, we compute $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}$ using the $\epsilon$-rule (see appendix A.6 for details) based on Equation (13):

$$R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}^{(j)} = \sum_k \frac{\mathbf{W}_{rn}^{(k,j)}\mathbf{h}_{t-1}^{(j)}}{\epsilon+\sum_i \mathbf{W}_{rn}^{(k,i)}\mathbf{h}_{t-1}^{(i)}} R_{\mathbf{n}_2}^{(k)}. \quad (15)$$

**(c) Compute $R_{\mathbf{h}_{t-1}}$:** $R_{\mathbf{h}_{t-1}}$ can be computed by adding Equations (10) and (15) together: $R_{\mathbf{h}_{t-1}} = R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t} + \sum_j R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}^{(j)}$. Notice that $R_{\hat{\mathbf{x}}_t}$ is the relevance of a node feature $\hat{\mathbf{x}}_t$, which is a row in $\hat{\mathbf{X}}_t$. By computing the set of relevances $\{R_{\hat{\mathbf{x}}_t^i}\}_{i=1}^N$ for all nodes, we can obtain the overall relevance matrix $R_{\hat{\mathbf{X}}_t}$, by concatenating the individual node relevances, i.e., $R_{\hat{\mathbf{X}}_t} = [R_{\hat{\mathbf{x}}_t^1}; R_{\hat{\mathbf{x}}_t^2}; \ldots; R_{\hat{\mathbf{x}}_t^N}]$.

| Dataset | Metric | SA | GNN-GI | GradCAM | GNNE | PGE | SubX | GCN-SE | T-GNNE | DyExplainer | Ours |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Reddit | Fidelity ↑ | 0.35 | 0.34 | 0.33 | 0.29 | 0.28 | 0.24 | 0.32 | 0.39 | 0.35 | **0.42** |
| | Fidelity+ ↑ | 0.19 | 0.23 | 0.22 | 0.16 | 0.12 | 0.10 | 0.21 | 0.24 | **0.27** | **0.27** |
| | Sparsity ↑ | 0.79 | 0.86 | 0.53 | 0.67 | 0.75 | 0.34 | 0.71 | 0.86 | 0.84 | **0.87** |
| | Stability ↓ | 0.29 | 0.17 | 0.26 | 0.25 | 0.27 | 0.30 | 0.21 | 0.15 | 0.18 | **0.13** |
| PeMS04 | Fidelity ↑ | 0.30 | 0.29 | 0.26 | 0.24 | 0.19 | 0.18 | 0.33 | **0.44** | 0.37 | 0.39 |
| | Fidelity+ ↑ | 0.21 | 0.19 | 0.17 | 0.16 | 0.13 | 0.14 | 0.24 | 0.25 | **0.30** | 0.29 |
| | Sparsity ↑ | **0.99** | **0.99** | 0.95 | 0.92 | 0.90 | 0.87 | 0.91 | 0.97 | 0.98 | **0.99** |
| | Stability ↓ | 0.18 | 0.22 | 0.25 | 0.22 | 0.23 | 0.27 | 0.23 | 0.17 | 0.19 | **0.15** |
| PeMS08 | Fidelity ↑ | 0.26 | 0.25 | 0.20 | 0.19 | 0.15 | 0.13 | 0.26 | 0.27 | 0.28 | **0.30** |
| | Fidelity+ ↑ | 0.19 | 0.16 | 0.12 | 0.11 | 0.09 | 0.08 | 0.20 | 0.21 | **0.26** | 0.25 |
| | Sparsity ↑ | 0.94 | 0.94 | **0.95** | 0.91 | 0.92 | 0.90 | 0.92 | 0.94 | 0.94 | **0.95** |
| | Stability ↓ | 0.15 | 0.16 | 0.18 | 0.14 | 0.15 | 0.23 | 0.16 | 0.13 | 0.16 | **0.12** |
| Enron | Fidelity ↑ | 0.20 | 0.19 | 0.16 | 0.09 | 0.09 | 0.08 | 0.19 | 0.21 | 0.19 | **0.23** |
| | Fidelity+ ↑ | 0.14 | 0.15 | 0.11 | 0.06 | 0.07 | 0.05 | 0.13 | 0.15 | 0.17 | **0.18** |
| | Sparsity ↑ | 0.84 | 0.83 | 0.79 | 0.75 | 0.74 | 0.70 | 0.83 | 0.81 | 0.82 | **0.85** |
| | Stability ↓ | 0.13 | 0.15 | 0.17 | 0.15 | 0.16 | 0.19 | **0.11** | 0.19 | 0.17 | 0.15 |
| FB | Fidelity ↑ | 0.29 | 0.22 | 0.19 | 0.16 | 0.15 | 0.10 | 0.33 | 0.31 | 0.33 | **0.36** |
| | Fidelity+ ↑ | 0.18 | 0.14 | 0.13 | 0.11 | 0.09 | 0.07 | 0.17 | 0.20 | 0.22 | **0.23** |
| | Sparsity ↑ | 0.94 | 0.93 | 0.91 | 0.90 | 0.86 | 0.80 | 0.92 | **0.98** | 0.95 | 0.96 |
| | Stability ↓ | 0.13 | 0.15 | 0.17 | 0.16 | 0.14 | 0.18 | 0.22 | 0.16 | 0.18 | **0.12** |
| COLAB | Fidelity ↑ | 0.50 | 0.45 | 0.39 | 0.27 | 0.26 | 0.25 | 0.43 | **0.55** | 0.51 | 0.53 |
| | Fidelity+ ↑ | 0.32 | 0.30 | 0.25 | 0.19 | 0.18 | 0.20 | 0.28 | 0.33 | 0.29 | **0.35** |
| | Sparsity ↑ | 0.96 | 0.95 | 0.94 | 0.93 | 0.93 | 0.90 | 0.94 | **0.99** | 0.96 | 0.96 |
| | Stability ↓ | 0.18 | 0.25 | 0.27 | **0.16** | 0.19 | 0.25 | 0.24 | 0.21 | 0.24 | 0.18 |

Table 1: Comparison with baselines on fidelity ($\tau_1 = 0.8$), fidelity+ ($\tau_1 = 0.8$), sparsity ($\tau_2 = 3! \times !10^{-4}$), and stability ($r = 20\%$). Compared methods: GNNExplainer (GNNE), PGExplainer (PGE), SubgraphX (SubX), T-GNNExplainer (T-GNNE), DyExplainer, and ours (`DGExplainer`). Best and second-best results are shown in **bold** and underline.

### 4.2 Stage 2: Relevance Back-Propagation in GCN

To find the relevance of the input data in a GCN, we start with the relevance of the output and backtrack through the network layers. We calculate the relevance of each layer's nodes using specific equations that distribute relevance from one layer to the previous. By repeating this process, we determine the relevance of the original input features. Finally, we average the absolute values of these relevances across all features to identify the importance of each node at a specific time. In the following, we show the concrete process in algorithm 1.

Then we backtrack in the GCN to get $R_{\mathbf{X}_t}$ from $R_{\hat{\mathbf{X}}_t}$. Note that the $R_{\hat{\mathbf{X}}_t}$ is the relevance of the output $\hat{\mathbf{X}}$ of the GCN at $t$ and $R_{\mathbf{F}_t^{(L)}} = R_{\hat{\mathbf{X}}_t}$. We rewrite Equation (1) as:

$$\mathbf{F}_t^{(l+1)} = \sigma(\mathbf{P}_t^{(l)}\mathbf{W}_t^{(l)}); \;\; \mathbf{P}_t^{(l)} := \mathbf{V}_t\mathbf{F}_t^{(l)}. \tag{16}$$

Let $(\mathbf{F}_t^{(l+1)})^{(k,:)}$, $(\mathbf{P}_t^{(l)})^{(k,:)}$, $(\mathbf{P}_t^{(l)})^{(:,k)}$, $(\mathbf{F}_t^{(l)})^{(:,k)}$ denote the $k$-th row of $\mathbf{F}_t^{(l+1)}$, the $k$-th row of $\mathbf{P}_t^{(l)}$, the $k$-th column of $\mathbf{P}_t^{(l)}$, the $k$-th column of $\mathbf{F}_t^{(l)}$, respectively. We have

$$(\mathbf{F}_t^{(l+1)})^{(k,:)} = \sigma((\mathbf{P}_t^{(l)})^{(k,:)}\mathbf{W}_t^{(l)}), \tag{17}$$

$$(\mathbf{P}_t^{(l)})^{(:,k)} := \mathbf{V}_t(\mathbf{F}_t^{(l)})^{(:,k)}. \tag{18}$$

Leveraging the $\epsilon$ rule, we assign the relevance by:

$$R_{(\mathbf{F}_t^{(l)})^{(j,k)}} = \sum_b \frac{\mathbf{V}^{(b,j)}(\mathbf{F}_t^{(l)})^{(j,k)}}{\epsilon + \sum_a \mathbf{V}_t^{(b,a)}(\mathbf{F}_t^{(l)})^{(a,k)}} R_{(\mathbf{P}_t^{(l)})^{(b,k)}}, \tag{19}$$

where $(\mathbf{W}_t^{(l)})^{(j,k)}$ represents the entry at the $j$-th row and $k$-th column of $\mathbf{W}_t^{(l)}$, and $\mathbf{V}_t^{(b,j)}$ denotes the entry at the $b$-th row and $j$-th column of $\mathbf{V}_t^{(k,j)}$. And the $R_{(\mathbf{P}_t^{(l)})^{(k,j)}}$ can be obtained similarly as $R_{(\mathbf{F}_t^{(l)})^{(j,k)}}$. The relevance $R_{\mathbf{F}_t^{(l)}}$ can be obtained from $R_{\mathbf{F}_t^{(l+1)}}$ using equation Equation (19), and $R_{(\mathbf{P}_t^{(l)})^{(k,j)}}$. Finally, the relevance $R_{\mathbf{F}_t^{(0)}}$ can be determined. Notice that $R_{\mathbf{F}_t^{(0)}} = R_{\mathbf{X}_t}$, so we have $R_{\mathbf{F}_t^{(0)}} = R_{\mathbf{X}_t}$, thus completing the backward process for obtaining relevance in the GCN. To further identify important nodes at a specific time step, we take the absolute values of the relevances and average them along the feature dimension to get the relevance of a node at time $t$: $R_{\mathbf{x}_t^i} = \sum_{j=1}^{D} |(R_{\mathbf{x}_t^i})^{(j)}|/D$.

Figure 3 illustrates the LRP process for a time step of these two states. Specifically, `DGExplainer` redistributes the relevance of the output hidden state, $R_{\mathbf{H}_t}$, to 1) the relevance of the input hidden state, $R_{\mathbf{H}_{t-1}}$, and 2) the relevance of the GCN-encoded feature, $R_{\hat{\mathbf{X}}_t}$. It then back-propagates the latter through the GCN and finally obtains the relevance of the input feature at this time step, $R_{\mathbf{X}_{t-1}^i}$. The entire algorithm is summarized in Appendix A.1.

## 5 Experiments

We conduct quantitative and qualitative experiments on six real-world graphs to answer the following research questions:
**RQ1**: Can the proposed `DGExplainer` learn high-quality explanations for the GCN-GRU model?
**RQ2**: What are the benefits of `DGExplainer` in explaining dynamic GNNs compared to static methods?
**RQ3**: How do the hyperparameters affect `DGExplainer`?

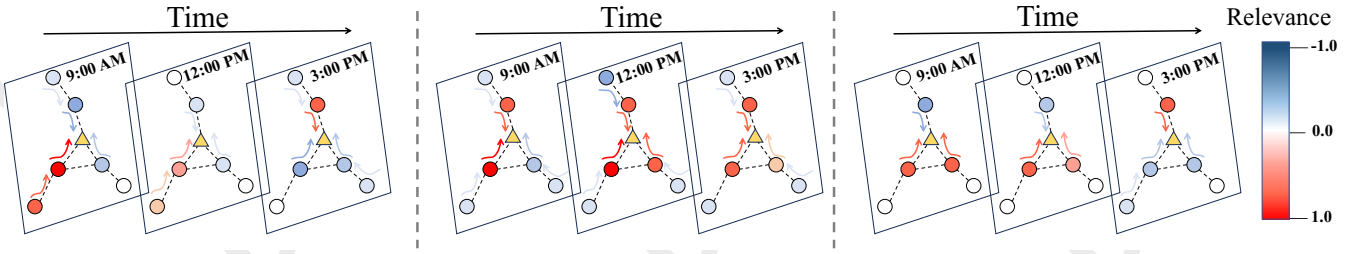



Figure 4: Illustration of the proposed method applied to the PeMS04 dataset. In this figure, warm colors indicate positive effects, while cold colors denote negative effects. The intensity of the color corresponds to the magnitude of the effect. From left to right, the subfigures represent the visualization results of GNN-GI, GNNExplainer, and the proposed method.

Implementation details are provided in Appendix A.4. Unless otherwise specified, we present the performance of `DGExplainer` on the GCN-GRU model in our experiments. Additionally, in Appendix A.7, we demonstrate the performance of `DGExplainer` across various other dynamic GNN models.

## 5.1 Experiment Settings

**Datasets.** We evaluate the proposed framework on six real-world datasets. For the link prediction tasks, we use four datasets: Reddit Hyperlink (Reddit) [Kumar *et al.*, 2018], Enron [Klimt and Yang, 2004], Facebook (FB) [Trivedi *et al.*, 2019], and COLAB [Rahman and Al Hasan, 2016]. For the node regression tasks, we use two datasets: PeMS04 and PeMS08 [Guo *et al.*, 2019][1]. The statistics of these datasets and the initial performance of GCN-GRU on them are presented in Appendix A.2.

**Baselines.** We assess our proposed method against eight baseline explanation methods. These include two general explanation methods: Sensitivity Analysis (SA) [Baldassarre and Azizpour, 2019] and GradCAM [Pope *et al.*, 2019]. Additionally, we compare our method with six GNN explanation methods: GNN-GI [Schnake *et al.*, 2020], GNNExplainer, PGExplainer, SubgraphX, GCN-SE, T-GNNExplainer, and DyExplainer. Detailed descriptions of these baseline methods are provided in Appendix A.3.

**Evaluation.** We compare the quality of each explanation baseline and our method using four quantitative metrics: confidence, sparsity, stability, and fidelity. Following the experimental setup of a previous work [Pareja *et al.*, 2020], we conduct experiments on link prediction and node classification. Detailed introduction of evaluation is in Appendix A.5.

## 5.2 Prediction and Explanation Performance

To address **RQ1**, we conducted a comprehensive comparison of our proposed method, `DGExplainer`, against several baseline methods. Our evaluation focused on two key aspects: prediction accuracy and the quality of explanations in identifying important nodes. The results demonstrate that `DGExplainer` outperforms the baselines in terms of fidelity and sparsity, providing more accurate and concise explanations. Additionally, our method exhibits good stability, ensuring consistent explanations even in the presence of minor perturbations, although on some datasets, it slightly underperforms SA and GradCAM. These results establish the effectiveness and reliability of our proposed method in capturing important nodes and providing reliable explanations in the context of link prediction and node regression tasks. Traditional explanation baselines also demonstrate competitive performance. For example, SA and GradCAM achieve strong sparsity, even surpassing some graph-specific explanation methods.

**Results on Fidelity, Fidelity+ and Sparsity.** Fidelity measures a method's ability to accurately capture important nodes. A high-fidelity explanation method is desirable. Fidelity+ is a surrogate version of fidelity, where a graph is sampled from the explanation subgraph by retaining each edge with probability $\alpha$ and erasing it with probability $1 - \alpha$. To assess fidelity, we ranked the nodes based on their importance and conducted occlusion experiments by selectively occluding a fraction of the top nodes while keeping $80\%$ of the nodes unchanged ($\tau_1 = 0.8$). For fidelity+, we define the explanation subgraph as the subgraph consisting of nodes with relevance greater than $\tau_1$. The proposed method consistently outperformed the baselines in terms of both fidelity and fidelity+, and sparsity across most datasets, as shown in Table 1. In the remaining datasets, our method achieved comparable results. The Fidelity+ gap between the proposed method and the baselines is larger than the Fidelity gap, further demonstrating DGExplainer's effectiveness in assigning higher relevance to important nodes.

**Results on Stability.** A stability evaluation was conducted to assess how well the explanation method handles perturbations in the input graph. We introduced random perturbations by adding additional edges to the original graph at a ratio of $r = 20\%$ and evaluated the resulting changes in the relevances generated by the model. A stable explanation method should provide consistent explanations when the input undergoes minor perturbations, resulting in lower stability scores. As presented in Table 1, our proposed method generally exhibited good stability, although it did not outperform SA and GNNExplainer. These findings indicate that our method demonstrates relative robustness to small perturbations in the input graph.

## 5.3 Qualitative Analysis

To address **RQ2**, we conducted quantitative experiments and visualizations of the generated explanations using

[1]pems.dot.ca.gov

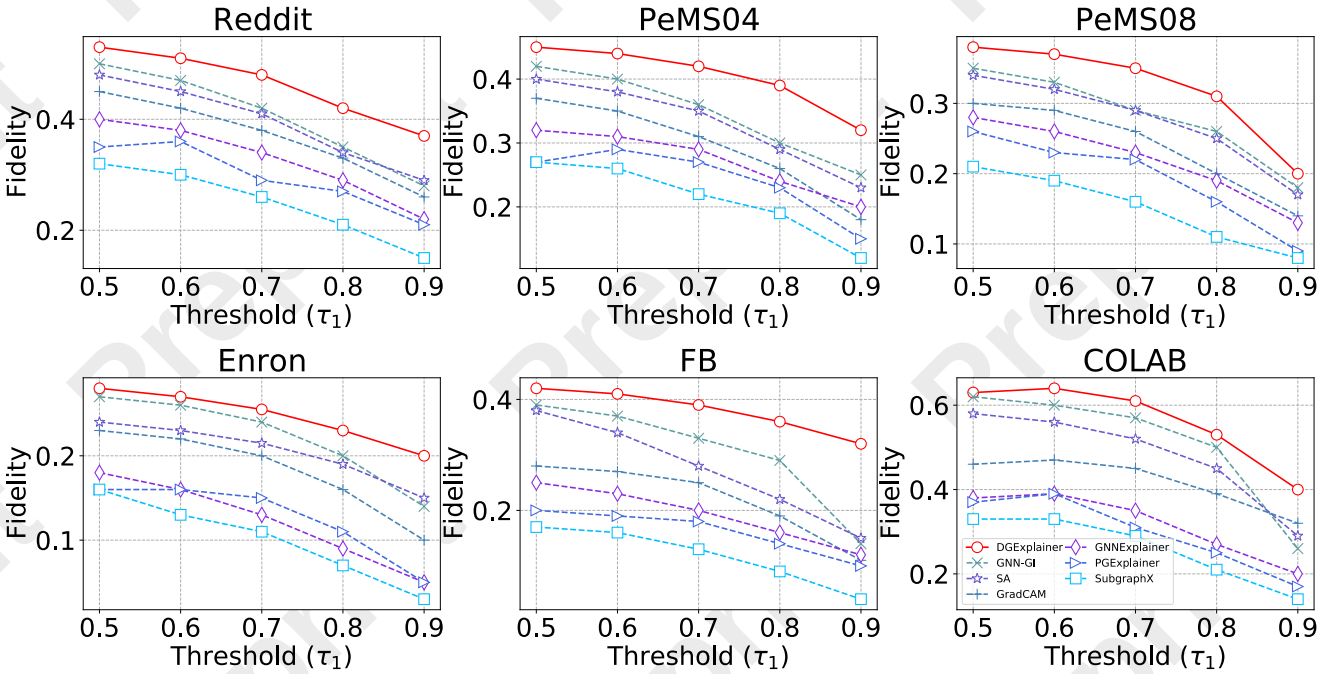



Figure 5: Comparison of different methods with the fidelity of similar levels of thresholds.

DGExplainer and baseline methods on the PeMS04 dataset, which represents traffic flow on a highway network. The results, presented in Figure 4, indicate that DGExplainer generates the most reasonable and detailed explanations compared to the GNN-GI and GNNExplainer approaches. Our analysis revealed several key findings: (a) GNN-GI tends to assign equally extreme relevances to every individual node, suggesting that each node has a strong correlation with the prediction. In contrast, GNNExplainer generates average scores for all the identified nodes. (b) GNN-GI identifies nearly all nodes as important, while GNNExplainer only identifies a few nodes as significant, disregarding the correlations of other nodes with the target variable.

These disparities in the visualization results are due to the fact that the comparison methods fail to capture the temporal patterns of dynamic graphs, treating each time step independently and considering only spatial information. In contrast, DGExplainer excels in generating comprehensive and context-aware explanations by effectively incorporating temporal dynamics into the analysis. By considering both spatial and temporal information, DGExplainer provides a more accurate understanding of the underlying relationships within the dynamic GNNs.

### 5.4 Parameter Sensitivity Analysis

To address **(RQ3)**, we investigate fidelity across various threshold values, denoted as $\tau_1 = \{0.5, 0.6, 0.7, 0.8, 0.9\}$. The fidelity analysis is presented in Figure 5. Our observations are as follows: (a) With smaller $\tau_1$ values, the fidelity is high. This is because a larger number of nodes are occluded when their relevance surpasses the threshold, resulting in a substantial change in accuracy. (b) As $\tau_1$ increases, the fidelity gradually decreases, with a steeper decline observed in the range of $[0.8, 0.9]$. Overall, our proposed method consistently achieves the highest fidelity across all thresholds and datasets, affirming the robustness of our framework. These findings provide substantial insights into the relationship between fidelity and the chosen threshold values, reinforcing the efficacy of our approach.

## 6 Conclusion

In this paper, we present DGExplainer, a novel and efficient framework that utilizes both layer-wise and time-wise relevance back-propagation to explain the predictions of dynamic Graph Neural Networks (GNNs). To evaluate DGExplainer's performance, we conduct both quantitative and qualitative experiments. The results demonstrate the framework's effectiveness in identifying crucial nodes for link prediction and node regression tasks, outperforming existing explanation methods. This research pioneers the exploration of dynamic GNNs, offering insights into their intricate structures, which is a significant challenge due to the complexity of inference in time-varying modules. Unlike existing static GNN explainers, DGExplainer does not require learning a surrogate function or executing any optimization procedures. Additionally, it can be extended to other advanced dynamic GNNs.

## Acknowledgements

Work in the paper is supported by NSF ECCS 2412484 and NSF RISE 2425748.

## References

[Bach *et al.*, 2015] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[Baldassarre and Azizpour, 2019] Federico Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. In *ICML Workshops*, 2019.

[Chen and Ying, 2023] Jialin Chen and Rex Ying. Tempme: Towards the explainability of temporal graph neural networks via motif discovery. *NeurIPS*, 36:29005–29028, 2023.

[Chen *et al.*, 2022] Jinyin Chen, Xueke Wang, and Xuanheng Xu. Gc-lstm: Graph convolution embedded lstm for dynamic network link prediction. *Applied Intelligence*, pages 1–16, 2022.

[Chen *et al.*, 2024] Yongqiang Chen, Yatao Bian, Bo Han, and James Cheng. How interpretable are interpretable graph neural networks? *arXiv preprint arXiv:2406.07955*, 2024.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[Duval and Malliaros, 2021] Alexandre Duval and Fragkiskos D Malliaros. Graphsvx: Shapley value explanations for graph neural networks. In *ECML PKDD*, pages 302–318. Springer, 2021.

[Fan *et al.*, 2021] Yucai Fan, Yuhang Yao, and Carlee Joe-Wong. Gcn-se: Attention as explainability for node classification in dynamic graphs. In *ICDM*, pages 1060–1065. IEEE, 2021.

[Goyal *et al.*, 2018] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv:1805.11273*, 2018.

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

[Gui *et al.*, 2020] Yihan Gui, Danshi Wang, Luyao Guan, and Min Zhang. Optical network traffic prediction based on graph convolutional neural networks. In *OECC*, pages 1–3. IEEE, 2020.

[Guo *et al.*, 2019] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*, pages 922–929, 2019.

[Hajiramezanali *et al.*, 2019] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. In *NeurIPS*, volume 32, 2019.

[Huang *et al.*, 2022] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks. *TKDE*, 2022.

[Kapoor *et al.*, 2020] Amol Kapoor, Xue Ben, Luyang Liu, Bryan Perozzi, Matt Barnes, Martin Blais, and Shawn O'Banion. Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv preprint arXiv:2007.03113*, 2020.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Klimt and Yang, 2004] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *ECML PKDD*, pages 217–226. Springer, 2004.

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

[Li *et al.*, 2017] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *CIKM*, pages 387–396, 2017.

[Luo *et al.*, 2020] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *NeurIPS*, 33:19620–19631, 2020.

[Ma *et al.*, 2020] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *SIGIR*, pages 719–728, 2020.

[Nguyen *et al.*, 2018] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW*, pages 969–976, 2018.

[Pareja *et al.*, 2020] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*, pages 5363–5370, 2020.

[Pope *et al.*, 2019] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *CVPR*, pages 10772–10781, 2019.

[Rahman and Al Hasan, 2016] Mahmudur Rahman and Mohammad Al Hasan. Link prediction in dynamic networks using graphlet. In *ECML PKDD*, pages 394–409. Springer, 2016.

[Sanchez-Lengeling *et al.*, 2020] Benjamin Sanchez-Lengeling, Jennifer Wei, Brian Lee, Emily Reif, Peter Wang, Wesley Qian, Kevin McCloskey, Lucy Colwell, and Alexander Wiltschko. Evaluating attribution for graph neural networks. *NeurIPS*, 33:5898–5910, 2020.

[Sankar *et al.*, 2018] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430*, 2018.

[Schlichtkrull *et al.*, 2020] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for nlp with differentiable edge masking. *arXiv preprint arXiv:2010.00577*, 2020.

[Schnake *et al.*, 2020] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T Schütt, Klaus-Robert Müller, and Grégoire Montavon. Higher-order explanations of graph neural networks via relevant walks. *arXiv preprint arXiv:2006.03589*, 2020.

[Schnake *et al.*, 2021] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T Schütt, Klaus-Robert Müller, and Grégoire Montavon. Higher-order explanations of graph neural networks via relevant walks. *TPAMI*, 44(11):7581–7596, 2021.

[Selvaraju *et al.*, 2017] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626, 2017.

[Seo *et al.*, 2018] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *ICONIP*, pages 362–373. Springer, 2018.

[Shapley, 1953] Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[Shrikumar *et al.*, 2017] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *ICML*, pages 3145–3153, 2017.

[Skarding *et al.*, 2021] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

[Trivedi *et al.*, 2019] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *ICLR*, 2019.

[Vu and Thai, 2020] Minh N Vu and My T Thai. PGM-explainer: Probabilistic graphical model explanations for graph neural networks. In *NeurIPS*, 2020.

[Wang *et al.*, 2023] Tianchun Wang, Dongsheng Luo, Wei Cheng, Haifeng Chen, and Xiang Zhang. Dyexplainer: Explainable dynamic graph neural networks. *arXiv preprint arXiv:2310.16375*, 2023.

[Xia *et al.*, 2022] Wenwen Xia, Mincai Lai, Caihua Shan, Yao Zhang, Xinnan Dai, Xiang Li, and Dongsheng Li. Explaining temporal graph models through an explorer-navigator framework. In *ICLR*, 2022.

[Xiong *et al.*, 2023] Ping Xiong, Thomas Schnake, Michael Gastegger, Grégoire Montavon, Klaus Robert Muller, and Shinichi Nakajima. Relevant walk search for explaining graph neural networks. In *ICML*, pages 38301–38324. PMLR, 2023.

[Yang *et al.*, 2020] Li Yang, Xiangxiang Gu, and Huaifeng Shi. A noval satellite network traffic prediction method based on gcn-gru. In *WCSP*, pages 718–723. IEEE, 2020.

[Ying *et al.*, 2019] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNNExplainer: Generating explanations for graph neural networks. In *NeurIPS*, volume 32, page 9240, 2019.

[You *et al.*, 2022] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *KDD*, pages 2358–2366, 2022.

[Yu *et al.*, 2018a] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *IJCAI*, pages 3634–3640, 2018.

[Yu *et al.*, 2018b] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *KDD*, pages 2672–2681, 2018.

[Yuan *et al.*, 2020] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *KDD*, pages 430–438, 2020.

[Yuan *et al.*, 2021] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In *ICML*, pages 12241–12252. PMLR, 2021.

[Zhang *et al.*, 2018] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *IJCV*, 126(10):1084–1102, 2018.

[Zhang *et al.*, 2022] Mengqi Zhang, Shu Wu, Xueli Yu, Qiang Liu, and Liang Wang. Dynamic graph neural networks for sequential recommendation. *TKDE*, 2022.

[Zhao *et al.*, 2018] Xujiang Zhao, Feng Chen, and Jin-Hee Cho. Deep learning for predicting dynamic uncertain opinions in network data. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1150–1155. IEEE, 2018.

[Zhao *et al.*, 2019] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *TITS*, 21(9):3848–3858, 2019.

[Zheng *et al.*, 2023] Xu Zheng, Farhad Shirani, Tianchun Wang, Wei Cheng, Zhuomin Chen, Haifeng Chen, Hua Wei, and Dongsheng Luo. Towards robust fidelity for evaluating explainability of graph neural networks. *arXiv preprint arXiv:2310.01820*, 2023.

[Zhu *et al.*, 2016] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *TKDE*, 28(10):2765–2777, 2016.