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

Anonymous authors
Paper under double-blind review

Abstract

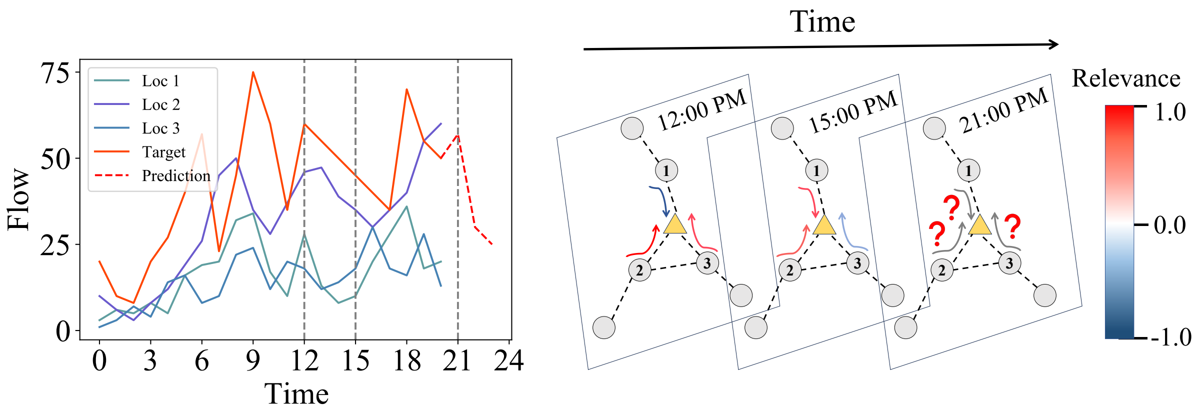
Dynamic graph neural networks (dynamic GNNs) have demonstrated remarkable effectiveness in analyzing time-varying graph-structured data. However, their black-box nature often hinders users from understanding their predictions, which can limit their applications. In recent years, there has been a surge in research aimed at explaining GNNs, but most studies have focused on static graphs, leaving the explanation of dynamic GNNs relatively unexplored. Explaining dynamic GNNs presents a unique challenge due to their complex spatial and temporal structures. As a result, existing approaches designed for explaining static graphs are not directly applicable to dynamic graphs because they ignore temporal dependencies among graph snapshots. To address this issue, we propose DGExplainer, which offers a reliable explanation of dynamic GNN predictions. DGExplainer utilizes the relevance back-propagation technique both time-wise and layer-wise. Specifically, it incorporates temporal information by computing the relevance of node representations along the inverse of the time evolution. Additionally, for each time step, it calculates layer-wise relevance from a graph-based module by redistributing the relevance of node representations along the back-propagation path. Quantitative and qualitative experimental results on six real-world datasets demonstrate the effectiveness of DGExplainer in identifying important nodes for link prediction and node regression in dynamic GNNs.

1 Introduction

Dynamic GNNs have achieved significant success in practical applications such as social network analysis (Zhu et al., 2016), transportation forecasting (Bui et al., 2022), pandemic forecasting (Kapoor et al., 2020), and recommender systems (Zhang et al., 2022). However, since most of the dynamic GNNs (Ma et al., 2020; Li et al., 2017; Nguyen et al.; Goyal et al., 2018; Yu et al., 2018a; Seo et al., 2018; Hajiramezanali et al., 2019) 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 trustfully employ dynamic GNN models, it is important to provide both accurate predictions and human-understandable explanations.

The explanation techniques for static GNNs have been extensively explored by recent studies. These techniques include approximation-based methods (Baldassarre & Azizpour, 2019; Pope et al., 2019b), 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 methods (Sundararajan et al., 2017; 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.1. However, these methods do not account for the unique temporal information essential for explaining dynamic GNNs. Directly applying existing explanation frameworks for static graphs to dynamic graphs is impractical, as it leads to discrete explanations for a graph sequence, with each graph snapshot being explained independently.

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



(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.

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 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. The complexity of dynamic graph data necessitates both temporal and spatial module designs in dynamic GNNs. This makes the explanation task challenging, as it requires identifying the influence of the input based on the output from these dynamic GNNs.

To integrate unique temporal and spatial information in explaining dynamic GNNs, we propose using layer-wise relevance propagation (LRP). Originally introduced by Bach et al. (2015) for image classifiers, LRP computes the relevance of each pixel in predicting an instance. Applying LRP to dynamic GNNs offers two key benefits. First, unlike most explanation techniques for static GNNs, it does not require learning a surrogate function or running any optimization procedure. Second, LRP evaluates the importance of sequences of edges or walks in the graph, rather than focusing solely on individual nodes or edges, making it particularly well-suited for explaining dynamic GNNs.

To address this challenge, we propose a framework called `DGExplainer` (Dynamic Graph Neural Network Explainer). The framework operates in three main steps. First, it decomposes the prediction of a dynamic GNN and computes the relevance in a time-related module using relevance back-propagation. Second, it calculates the relevance of the input features by back-propagating through the graph-related modules (e.g., a GCN module) layer by layer at each time step. Finally, by aggregating the relevance from the previous steps, we obtain the final relevance of node features, which represents their importance to the prediction. The contributions of our work are as follows:

- This work aims to explain the predictions of dynamic graph neural networks, marking one of the pioneering efforts to tackle this challenge.
- We propose a novel framework, `DGExplainer`, designed to generate explanations for dynamic GNNs from a decomposition perspective. `DGExplainer` effectively calculates relevances that represent the contributions of each component in a dynamic graph.
- We demonstrate the effectiveness of `DGExplainer` on six real-world datasets. Quantitative experiments across three evaluation metrics show that our method provides faithful explanations. Furthermore, qualitative experiments demonstrate that `DGExplainer` offers significant advantages over other baseline methods in effectively explaining dynamic GNNs.

2 Problem Definition

Given a dynamic graph as a sequence of snapshots $\mathcal{G} = \{\mathcal{G}_t\}_{t=1}^{T}$, where T is the length of the sequence. $\mathcal{G}_t = \{\mathcal{V}_t, \mathcal{E}_t\}$ represents the graph at time t and $\mathcal{V}_t$, $\mathcal{E}_t$ represents the node set, the edge set, respectively. The adjacency matrix at time step t is represented as $\mathbf{A}_t \in \mathbb{R}^{N\times N}$, where $N = |\mathcal{V}_t|$ is the number of nodes. The feature matrix is denoted as $\mathbf{X}_t \in \mathbb{R}^{N\times D}$, where D is the feature dimension, and $\mathbf{x}_t^i = (\mathbf{X}_t^{(i,:)})^\top \in \mathbb{R}^D$ is the attribute vector for node i at time step t, *i.e.* the i-th row of $\mathbf{X}_t$. Without loss of generality, here $\mathbf{A}^{(i,j)}$ denotes the entry at i-th row, j-th column of adjacency matrix $\mathbf{A}$, and $\mathbf{x}^{(i)}$ denotes the i-th entry of vector $\mathbf{x}$. R_k represents the relevance of k, where k can be a node, an edge, a feature, etc. Also, $R_{k_1\leftarrow k_2}$ denotes the relevance of k_1 is distributed from k_2. The goal of explaining dynamic GNNs is to find the subgraph in $\mathcal{G}$, i.e., nodes and edges, that is the most important at time step t, given a dynamic GNN model $f(\mathcal{G})$.

3 Explaining dynamic GNNs via `DGExplainer`

In this section, we first provide an overview of dynamic graph neural networks in Section 3.1. We then introduce the GCN-GRU model, which will be used later to demonstrate our explanation method. Next, we describe the layer-wise relevance propagation technique in Section 3.2. Finally, we elaborate on the proposed method, `DGExplainer`, in Section 3.3, which explains dynamic GNNs by back-propagating relevance through both the time-varying and message-passing paths to the inputs.

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. Numerous dynamic GNNs have been proposed for modeling dynamic graphs (Goyal et al., 2018; Yu et al., 2018a; Seo et al., 2018; Hajiramezanali et al., 2019). A notable approach co-trains a GNN with a recurrent neural network (RNN), referred to as a GNN-RNN model, such as GCN-GRU (Zhao et al., 2019), ChebNet-LSTM (Seo et al., 2018), and GCN-RNN (Pareja et al., 2020). In terms of performance, recent methods 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. The GCN-GRU model has wide applications (Gui et al., 2020; Yang et al., 2020; Zhao et al., 2018). For example, in traffic flow prediction, the GNN models the dynamics of traffic as an information dissemination process, while the RNN captures the spatial dependency. A detailed introduction to the related research can be found in Appendix A.1. Besides explaining the GCN-GRU model, we also consider applying `DGExplainer` to other dynamic GNNs with different GNN or RNN architectures. Additional experimental results can be found in Appendix A.6.

The GCN-GRU model: The structure of the GCN-GRU model is summarized in Figure 2. The GCN module encodes node dependencies at each time step using a graphical representation and outputs the node representations to the GRU module, which captures temporal dependencies across different time steps. The following outlines the forward process of the GCN-GRU model.

(a) The Graph Convolutional Network (GCN) module: In the GCN-GRU model, the GCN represents a node using local information from its surrounding neighbors (Kipf & 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 node representations from all time steps $\{\hat{\mathbf{X}}_t\}_{t=1}^{T}$ obtained from the GCN are then fed into a GRU.

(b) The Gated Recurrent Unit (GRU) module: The GRU is a variant of the RNN designed to learn long-term dependencies using two selective gates (Cho et al., 2014). In the GCN-GRU model, the GRU

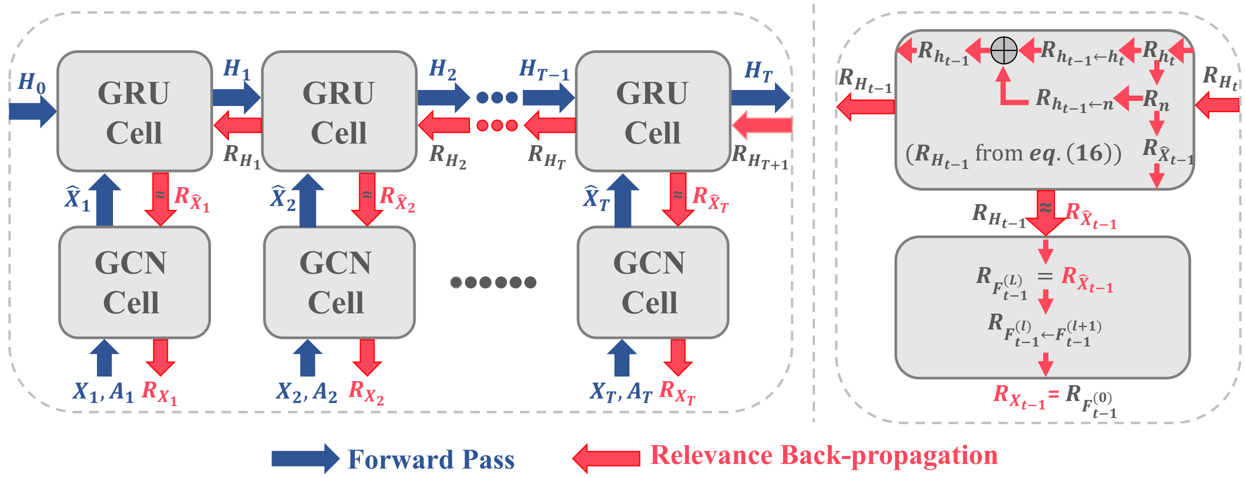


Figure 2: Left: The network structure of the GCN-GRU model and the back-propagation of the relevances. Note that the GRU cells and GCN cells share the same parameters. $\{\mathbf{H}_t\}_{t=0}^T$, $\{\mathbf{X}_t\}_{t=1}^T$, $\{\hat{\mathbf{X}}_t\}_{t=1}^T$, $\{\mathbf{A}_t\}_{t=1}^T$ represent the hidden features in GRU, node features, transformed features by GCN, and adjacency matrices at different time steps, respectively. Right: An illustration of `DGExplainer` for calculating relevances in a backward manner. The feature relevance is computed by first back-propagating the final output $R_{\mathbf{h}_T}$ through the GRU and then through the GCN.

module captures dependencies across different time steps through gate units trained to manage inputs and memory states, enabling the retention of information over longer periods (Zhao et al., 2018). 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:

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

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

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

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

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

3.2 Layer-wise Relevance Propagation

Layer-wise relevance propagation (LRP) (Bach et al., 2015) is a technique for explaining the predictions of deep neural networks. It operates on the assumption that a neuron's relevance is proportional to its weighted activation value. This follows the intuition that a larger output activation indicates that the neuron carries more information and contributes more significantly to the final result.

The concept behind LRP assumes that the relevance, denoted as $R_{k_2}^{(l+1)}$, is known for a neuron in the subsequent layer $(l+1)$. This assumption allows us to break down and distribute this relevance to the neurons, denoted as k_1, in the current layer l that contribute input to the neuron k_2. This process enables us to determine the relevance value for a neuron k_1 in layer l by aggregating all the incoming messages from neurons in layer $(l+1)$. A notable challenge in LRP is formulating an appropriate rule for redistributing relevance across each layer. Drawing insights from prior studies (Bach et al., 2015; Binder et al., 2016; Schnake et al., 2021), we describe the propagation rule as follows:

$$R_{k_1 \leftarrow k_2}^{(l,l+1)} = \sum_{k_2} \frac{\mathbf{W}_{k_1 k_2} a_{k_1}^{(l)}}{\epsilon + \sum_{k_1} \mathbf{W}_{k_1 k_2} a_{k_1}^{(l)}} R_{k_2}^{(l+1)}, \tag{3}$$

where $\mathbf{W}_{k_1k_2}$ represents the connection weight between neurons k_1 and k_2. $R_{k_2}^{(l+1)}$ is the relevance for neuron k_2 at layer $(l+1)$, and $R_{k_1\leftarrow k_2}^{(l,l+1)}$ is the relevance for neuron k_1 derived from k_2 at layer l. $a_{k_1}^{(l)}$ denotes the activation of neuron k_1 at layer l. The term ϵ is a predefined stabilizer that prevents the denominator from being zero. Clearly, the connection between the relevance and the weighted activation $\mathbf{W}_{k_1k_2}a_{k_1}^{(l)}$ is discernible. This relationship indicates that the relevance varies in proportion to the magnitude of the weighted activation. Additionally, the nature of the contribution, whether positive or negative, depends on the sign of the weighted activation. The proof of Equation (3) is provided in Appendix A.5.

3.3 The Proposed `DGExplainer` for Explaining Dynamic Graphs

We propose the `DGExplainer` method for explaining dynamic GNNs using a relevance back-propagation process. Similar to many recent backward-based methods (Schnake et al., 2021; Bach et al., 2015; Pope et al., 2019a), `DGExplainer` aims to identify the most important subset of node features that contribute to the prediction. Specifically, it calculates relevances within the range of $(-1,1)$ to determine the extent to which each component of the model influences the prediction.

We elaborate on the `DGExplainer` framework in Figure 2. `DGExplainer` redistributes the final prediction $f(\cdot)$, represented as $\mathbf{H}_{T+1}$, to the relevance of the node representation $\mathbf{H}_T$ at the last time step. This redistribution process is repeated for each time step and its preceding time step, ultimately obtaining the relevances corresponding to the inputs, i.e., $R_{\mathbf{X}_1}$. The blue arrows and letters indicate the forward propagation and the parameters passed along the path, while the red ones represent the LRP process and the computed relevances of the input features. The right figure illustrates the LRP process at one timeslot, where `DGExplainer` redistributes the relevance $R_{\mathbf{H}_t}$ of the representation $\mathbf{H}_t$ to the relevances of each neuron in this layer, and finally obtains the relevance $R_{\mathbf{H}_{t-1}}$.

Algorithm 1 `DGExplainer`

Input: The input $\{\mathbf{X}_t\}_{t=1}^T$ and $\{\mathbf{A}_t\}_{t=1}^T$, the final relevance $\{R_{\mathbf{h}_T^j}\}_{j=1}^N$, the pre-trained model $f(\cdot)$.
Output: The relevances $\{R_{\mathbf{X}_t}\}_{t=1}^T$

```
// Forward process:
for each t ∈ [1, T] do
    Compute X̂_t via F_t^(l+1) = σ(V_t F_t^(l) W_t^(l)) with F_t^(0) = X_t, F_t^(L) = X̂_t.
    for each j ∈ [1, N] do
        Compute the hidden state h_t for the j-th sample
        X̂_t^(j,:) via Equation (2) with h_{t-1}.
    end for
end for
// Backward process:
for each t = T, T − 1, . . . , 1 do
    for each j ∈ [1, N] do
        Compute R_n, R_n1, R_n2 via Equations (7), (12) and (13), R_{h_{t-1}} via Equations (8), (15) and (16), and R_{x̂_t}
    for the j-th sample X̂_t^(j,:) via Equation (14) and hence obtain R_{x̂_t^j}.
    end for
    Stack {R_{x̂_t^j}}_{j=1}^N to get R_{X̂_t}.
    Calculate R_{X_t} by iteratively applying Equations (20) and (21) with R_{X̂_t} = R_{F_t^(L)} and R_{X_t}.
end for
return {R_{X_t}}_{t=1}^T.
```

3.3.1 Compute the Relevances in GRU

Given $R_{\mathbf{h}_T}$ at $t=T$, the goal is to compute $R_{\mathbf{h}_{t-1}}$ and $R_{\hat{\mathbf{x}}_{t-1}}$ from $R_{\mathbf{h}_t}$. As described in Section 3.2, 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:

$$R_{\mathbf{h}_{t-1}} = R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t} + R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}} + R_{\mathbf{h}_{t-1}\leftarrow\mathbf{z}} + R_{\mathbf{h}_{t-1}\leftarrow\mathbf{r}}. \tag{4}$$

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:

$$R_{\mathbf{h}_t} = R_{\mathbf{h}_{t-1}} + R_{\mathbf{n}}, \tag{5}$$

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

$$\frac{R_{\mathbf{h}_{t-1}}}{R_{\mathbf{n}}} = \frac{a_{\mathbf{h}_{t-1}}}{a_{\mathbf{n}}} = \frac{\mathbf{z}\odot\mathbf{n}}{(1-\mathbf{z})\odot\mathbf{h}_{t-1}}. \tag{6}$$

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 (5) and (6) obtains:

$$R_{\mathbf{h}_t\leftarrow\mathbf{n}} = \frac{\mathbf{z}\odot\mathbf{n}}{\mathbf{h}_t+\epsilon}\odot R_{\mathbf{h}_t}, \tag{7}$$

$$R_{\mathbf{h}_t\leftarrow\mathbf{h}_{t-1}} = \frac{(1-\mathbf{z})\odot\mathbf{h}_{t-1}}{\mathbf{h}_t+\epsilon}\odot R_{\mathbf{h}_t}, \tag{8}$$

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 can calculate:

$$\mathbf{n}_1 := \mathbf{W}_{in}\hat{\mathbf{x}}_t, \tag{9a}$$

$$\mathbf{n}_2 := \mathbf{r}\odot(\mathbf{W}_{hn}\mathbf{h}_{t-1}) = \mathbf{W}_{rn}\mathbf{h}_{t-1}, \tag{9b}$$

$$\mathbf{b}_{\mathbf{n}} := \mathbf{b}_{in} + \mathbf{r}\odot\mathbf{b}_{hn}. \tag{9c}$$

Then their relevance satisfies:

$$R_{\mathbf{n}} = R_{\mathbf{n}_1} + R_{\mathbf{n}_2} + R_{\mathbf{b}_n}, \tag{10}$$

$$R_{\mathbf{n}_1} : R_{\mathbf{n}_2} : R_{\mathbf{b}_n} = \mathbf{n}_1 : \mathbf{n}_2 : \mathbf{b}_n. \tag{11}$$

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

$$R_{\mathbf{n}_1} = \frac{\mathbf{W}_{in}\hat{\mathbf{x}}_t}{\epsilon + (\mathbf{W}_{in}\hat{\mathbf{x}}_t + \mathbf{b}_{in} + \mathbf{r}\odot(\mathbf{W}_{hn}\mathbf{h}_{t-1} + \mathbf{b}_{hn}))}\odot R_{\mathbf{n}}, \tag{12}$$

$$R_{\mathbf{n}_2} = \frac{\mathbf{r}\odot(\mathbf{W}_{hn}\mathbf{h}_{t-1})}{\epsilon + (\mathbf{W}_{in}\hat{\mathbf{x}}_t + \mathbf{b}_{in} + \mathbf{r}\odot(\mathbf{W}_{hn}\mathbf{h}_{t-1} + \mathbf{b}_{hn}))}\odot R_{\mathbf{n}}. \tag{13}$$

Let $\mathbf{n}_1^{(k)}$ denote the k-th entry of $\mathbf{n}_1$, according to Equation (9a), we have $\mathbf{n}_1^{(k)} = \sum_j \mathbf{W}_{in}^{(k,j)}\hat{\mathbf{x}}_t^{(j)}$. The relevance $R_{\mathbf{n}_1}$ is redistributed in proportion to the contribution for $\mathbf{n}_1$ and hence $R_{\hat{\mathbf{x}}_t}$, which equals to $R_{\hat{\mathbf{x}}_t\leftarrow\mathbf{n}_1}$ because $\mathbf{n_1}$ is the only source of the relevance to $\hat{\mathbf{x}}_t$ by using LRP-ϵ rule (Bach et al., 2015):

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

Since $\mathbf{h}_{t-1}$ only influences $\mathbf{n}_2$ among the three parts of $\mathbf{n}$, we obtain $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}$ using ϵ-rule for Equation (9b):

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

(c) Compute $R_{\mathbf{h}_{t-1}}$: Upon obtaining $R_{\mathbf{n}\leftarrow\mathbf{h_t}}$, and $R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h_t}}$ in Equations (7) and (8), based on Equation (10), $R_{\mathbf{h}_{t-1}}$ can be computed by adding Equations (8) and (15) together:

$$R_{\mathbf{h}_{t-1}} = R_{\mathbf{h}_{t-1}\leftarrow\mathbf{h}_t} + \sum_j R^{(j)}_{\mathbf{h}_{t-1}\leftarrow\mathbf{n}}. \tag{16}$$

Notice that $R_{\hat{\mathbf{x}}_t}$ is the relevance of a sample $\hat{\mathbf{x}}_t$, which is a row in $\hat{\mathbf{X}}_t$. By computing $\{R_{\hat{\mathbf{x}}^i_t}\}^N_{i=1}$, we can get $R_{\hat{\mathbf{X}}_t} = [R_{\hat{\mathbf{x}}^1_t}; R_{\hat{\mathbf{x}}^2_t}; \ldots; R_{\hat{\mathbf{x}}^N_t}]$.

3.3.2 Back-Propagate the Relevances in GCN

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 the time step t and $R_{\mathbf{F}^{(L)}_t} = R_{\hat{\mathbf{X}}_t}$. We can rewrite Equation (1) as:

$$\mathbf{F}^{(l+1)}_t = \sigma(\mathbf{P}^{(l)}_t\mathbf{W}^{(l)}_t); \quad \mathbf{P}^{(l)}_t := \mathbf{V}_t\mathbf{F}^{(l)}_t. \tag{17}$$

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

$$(\mathbf{F}^{(l+1)}_t)^{(k,:)} = \sigma((\mathbf{P}^{(l)}_t)^{(k,:)}\mathbf{W}^{(l)}_t), \tag{18}$$

$$(\mathbf{P}^{(l)}_t)^{(:,k)} := \mathbf{V}_t(\mathbf{F}^{(l)}_t)^{(:,k)}. \tag{19}$$

Leveraging the ϵ rule, we assign the relevance by:

$$R_{(\mathbf{P}^{(l)}_t)^{(k,j)}} = \sum_b \frac{(\mathbf{P}^{(l)}_t)^{(k,j)}(\mathbf{W}^{(l)}_t)^{(j,b)}}{\epsilon + \sum_i(\mathbf{P}^{(l)}_t)^{(k,i)}(\mathbf{W}^{(l)}_t)^{(i,b)}} R_{(\mathbf{F}^{(l+1)}_t)^{(k,b)}}, \tag{20}$$

$$R_{(\mathbf{F}^{(l)}_t)^{(j,k)}} = \sum_b \frac{\mathbf{V}^{(b,j)}(\mathbf{F}^{(l)}_t)^{(j,k)}}{\epsilon + \sum_a \mathbf{V}^{(b,a)}_t(\mathbf{F}^{(l)}_t)^{(a,k)}} R_{(\mathbf{P}^{(l)}_t)^{(b,k)}}, \tag{21}$$

where $(\mathbf{W}^{(l)}_t)^{(j,k)}$ represents the entry at the j-th row and k-th column of $\mathbf{W}^{(l)}_t$, and $\mathbf{V}^{(b,j)}_t$ denotes the entry at the b-th row and j-th column of $\mathbf{V}^{(k,j)}_t$. The relevance $R_{\mathbf{F}^{(l)}_t}$ can be obtained from $R_{\mathbf{F}^{(l+1)}_t}$ using equations Equations (20) and (21). Finally, the relevance $R_{\mathbf{F}^{(0)}_t}$ can be determined. Notice that $R_{\mathbf{F}^{(0)}_t} = R_{\mathbf{X}_t}$, so we have $R_{\mathbf{F}^{(0)}_t} = R_{\mathbf{X}_t}$, 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: $R_{\mathbf{x}^i_t} = \sum^D_{j=1} |(R_{\mathbf{x}^i_t})^{(j)}|/D$. The entire algorithm is summarized in Algorithm 1.

4 Experiments

We conduct quantitative and qualitative experiments on six real-world graphs to address 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`?

Unless otherwise specified, we present the performance of `DGExplainer` on the GCN-GRU model in our experiments. Additionally, in Appendix A.6, we demonstrate the performance of `DGExplainer` across various other dynamic GNN models.

Table 1: Comparison with baseline methods in terms of fidelity ($\tau_1 = 0.8$), sparsity ($\tau_2 = 3 \times 10^{-4}$), and stability ($r = 20\%$). The methods compared are GNNExplainer (GNNE), PGExplainer (PGE), SubgraphX (SubX), and T-GNNExplainer (T-GNNE). 'Ours' refers to `DGExplainer`.

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

4.1 Datasets and Baselines

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 & Yang, 2004), Facebook (FB) (Trivedi et al., 2019), and COLAB (Rahman & 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: (a) Sensitivity Analysis (SA) and (b) GradCAM. Additionally, we compare our method with six GNN explanation methods: (c) GNN-GI, (d) GNNExplainer, (e) PGExplainer, (f) SubgraphX, (g) GCN-SE, and (h) T-GNNExplainer. Detailed descriptions of these baseline methods are provided in Appendix A.3.

4.2 Experiment Settings

Evaluation. We compare the quality of each explanation baseline and our proposed method using four quantitative metrics: confidence, sparsity, stability, and fidelity. Details of these evaluation metrics are elaborated in Appendix A.4. Following the experimental setup of a previous work (Pareja et al., 2020), we conduct experiments on link prediction and node classification.

- **Link prediction**: For this task, we concatenate the feature embeddings of nodes u and v as $[(\mathbf{h}_T^u)^\top; (\mathbf{h}_T^v)^\top]^\top$ and use a multi-layer perceptron (MLP) to predict the link probability by optimizing the cross-entropy loss. We experiment with the Reddit, Enron, FB, and COLAB datasets and use the Area Under the Curve (AUC) as the evaluation metric.
- **Node regression**: To predict the value for a node u at time t, we apply a softmax activation function to the last layer of the GCN, resulting in the probability vector $\mathbf{h}_t^u$. We use the PeMS04 and PeMS08 datasets for this task and evaluate the performance using the mean absolute error (MAE) metric.

[1] pems.dot.ca.gov

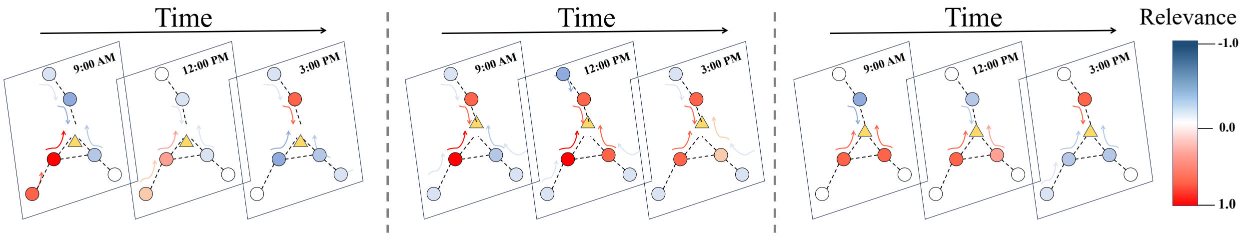


Figure 3: Illustration of the proposed method applied to the PeMS04 dataset. In this figure, warm colors indicate positive effects, while cold colors denote negative effects. The intensity of the color corresponds to the magnitude of the effect. From left to right, the subfigures represent the visualization results of GNN-GI, GNNExplainer, and the proposed method.

Implementation Details. We conducted all our experiments on a Linux machine equipped with four NVIDIA RTX A4000 Ti GPUs, each with 16GB of RAM. We used a two-layer GCN and trained the model for 1000 epochs using the Adam optimizer (Kingma & Ba, 2014), with an initial learning rate of 0.01. For the link prediction task, we employed a two-layer MLP with 64 hidden units. We tested the stabilizer ϵ with values $\{1e\text{-}5, 1e\text{-}4, 1e\text{-}3, 1e\text{-}2, 1e\text{-}1, 1, 2\}$. In stability experiments, we set r to $\{5\%, 10\%, 15\%, 20\%, 30\%\}$. The model performance results are based on the average analysis of 10 runs. The output embedding of a node u produced by the GCN-GRU model at time t is represented by $\mathbf{h}_t^u$.

4.3 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.

Results on fidelity and sparsity. Fidelity measures a method's ability to accurately capture important nodes. A high-fidelity explanation method is desirable. 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 ($\tau_1 = 0.8$). The proposed method consistently outperformed the baselines in terms of both fidelity and sparsity across most datasets, as shown in Table 1. In the remaining datasets, our method achieved comparable results.

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.

4.4 Qualitative Analysis

To address **RQ2**, we conducted quantitative experiments and visualizations of the generated explanations using `DGExplainer` and baseline methods on the PeMS04 dataset, which represents traffic flow on a highway network. The results, presented in Figure 3, 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

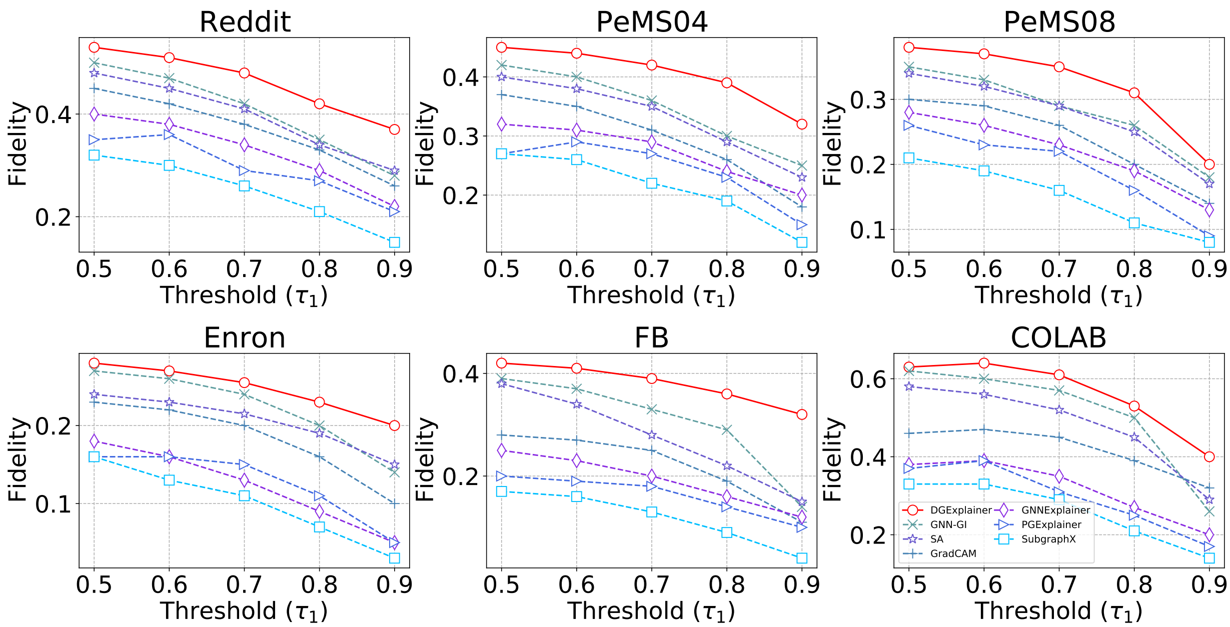


Figure 4: Comparison of different methods with the fidelity of similar levels of thresholds.

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.

4.5 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 4. Our observations are as follows: (a) With smaller τ_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 τ_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.

5 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 holds promise for extension to other advanced dynamic GNNs.

References

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.

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

Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. In *ICANN*, pp. 63–71, 2016.

Khac-Hoai Nam Bui, Jiho Cho, and Hongsuk Yi. Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues. *Applied Intelligence*, 52(3):2763–2774, 2022.

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

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.

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

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

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

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

Yihan Gui, Danshi Wang, Luyao Guan, and Min Zhang. Optical network traffic prediction based on graph convolutional neural networks. In *2020 Opto-Electronics and Communications Conference (OECC)*, pp. 1–3. IEEE, 2020.

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

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

Bo Kang, Jefrey Lijffijt, and Tijl De Bie. Explaine: An approach for explaining network embedding-based link predictions. *arXiv preprint arXiv:1904.12694*, 2019.

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.

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

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

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

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

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

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.

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

Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW Companion*.

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*, pp. 5363–5370, 2020.

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

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

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

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 'Why should I trust you?' explaining the predictions of any classifier. In *KDD*, pp. 1135–1144, 2016.

Benjamin Sanchez-Lengeling, Jennifer Wei, Brian Lee, Emily Reif, Peter Wang, Wesley Wei Qian, Kevin McCloskey, Lucy Colwell, and Alexander Wiltschko. Evaluating attribution for graph neural networks. In *NeurIPS*, volume 33, 2020.

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.

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.

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*, pp. 618–626, 2017.

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

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

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *ICML*, pp. 3145–3153, 2017.

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.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *ICML*, pp. 3319–3328. PMLR, 2017.

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

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

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.

Li Yang, Xiangxiang Gu, and Huaifeng Shi. A noval satellite network traffic prediction method based on gcn-gru. In *2020 International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 718–723. IEEE, 2020.

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

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

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

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*, pp. 2672–2681, 2018b.

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

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

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. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858, 2019.

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)*, pp. 1150–1155. IEEE, 2018.

Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(10):2765–2777, 2016.

A Appendix

In this appendix, we present related work, provide a more detailed introduction to the datasets, describe the evaluation metrics for baselines, explain the LRP method in detail, and show additional experiments to demonstrate the superiority of the proposed method, `DGExplainer`.

A.1 Related Work

We review previous studies related to our work, focusing first on the recent advances in dynamic graph neural networks and then on existing explainability methods for static GNNs.

Dynamic graph neural networks Dynamic graph neural networks (Dynamic GNNs) consider both temporal and graph-structural information to tackle dynamic graphs. These networks are commonly applied in social media, citation networks, transportation networks, and pandemic networks. DANE (Li et al., 2017) is an efficient dynamic GNN that updates node embeddings using eigenvectors of the graph's Laplacian matrix based on the graph from the previous time step. CTDANE (Nguyen et al.) and NetWalk (Yu et al., 2018b) extend random walk-based approaches by enforcing temporal rules on the walks. Additionally, there are embedding methods that aggregate neighboring node features. For example, DynGEM (Goyal et al., 2018) and Dyngraph2vec (Goyal et al., 2020) use deep autoencoders to encode snapshots of dynamic graphs.

A prevalent category of approaches combines GNNs with recurrent architectures, where the GNNs extract graph-structural information, and the recurrent units handle dynamic flows. GCRN (Seo et al., 2018) leverages GCN layers to obtain node embeddings and feeds them into recurrent layers to track the dynamism. STGCN (Yu et al., 2018a), stacked with ST-Conv blocks, proposes a sophisticated architecture that effectively captures complex localized spatial-temporal correlations. Instead of directly integrating RNNs into the entire structure, EvolveGCN (Pareja et al., 2020) uses RNNs to update the weights of GCNs. Another approach (Hajiramezanali et al., 2019) introduces variational autoencoder versions for dynamic graphs, VGRNN and SI-VGRNN. Both models use a GCN integrated into an RNN as an encoder to track the temporal evolution of the graph. Despite these advances, recent methods still do not consistently outperform the GCN-GRU model (Pareja et al., 2020). For these reasons, we choose to use the GCN-GRU model in this work.

Explainability on GNNs Although there are currently no established explainability approaches for dynamic GNNs, methods for interpreting other types of GNNs exist and can be categorized into two main directions. The first direction focuses on generic model-agnostic explanation methods that consider black-box models. These methods typically accumulate local effects and learn a locally faithful approximation, such as local methods and partial dependence plots (Friedman, 2001). Examples include Shapley values (Shapley, 1953) and LIME (Ribeiro et al., 2016). The second direction focuses on the specific structure of neural networks, uncovering important components in the computation through feature gradients, relevance scores, and counterfactual reasoning (Kang et al., 2019). However, these methods do not consider the graph's structural or temporal information, which is crucial for the success of dynamic GNNs. Recently, a few explanation methods specialized for GNNs have emerged, such as GNNExplainer (Ying et al., 2019), PGM-Explainer (Vu & Thai, 2020), and PGExplainer (Luo et al., 2020).

However, existing explainability methods are primarily designed for static graphs and do not account for the dynamic nature of graphs. To address this gap, we propose `DGExplainer`, a framework that provides faithful explanations for dynamic GNNs. Our approach decomposes the prediction of a dynamic GNN and computes the relevances in a time-related module using Layer-wise Relevance Propagation (LRP). We then further compute the relevances of the input features by back-propagating the relevances through the graph-related modules at each timestamp, considering both the graph's structural and temporal information. This method enables us to generate more accurate explanations for dynamic GNNs.

Despite the success of existing explainability methods, they primarily focus on static graphs and overlook the temporal or dynamic aspects of graphs. This limitation has spurred the development of explainability methods for dynamic GNNs, such as our proposed framework `DGExplainer`. `DGExplainer` employs backward propagation to compute the relevance of each input feature in the dynamic GNN model, taking into account both the graph's structural and temporal information.

A.2 Datasets

The statistics of the datasets and the initial performance of GCN-GRU on these datasets are summarized in Table 2.

Table 2: Dataset statistics and performance of the GCN-GRU model. For performance metrics, we report AUC (%) on the Reddit, Enron, FB, and COLAB datasets, and MAE on the PeMS04 and PeMS08 datasets.

Dataset	Reddit	PeMS04	PeMS08	Enron	FB	COLAB
# Nodes	55,863	307	170	184	663	315
# Edges	858,490	680	340	266	1068	308
# Train/Test	122/34	45/14	50/12	8/3	6/3	7/3
# Time Step	6	4	4	4	4	4
Performance	0.702	55.29	59.35	0.951	0.870	0.879

- **Reddit** is a directed network extracted from posts that generate hyperlinks connecting one subreddit to another. It includes various features, such as the source post, target URL, post title, and comment text, along with metadata like the number of upvotes and downvotes each post and comment received. The Reddit Hyperlink dataset comprises hyperlink information from over 3 million posts and their associated comments on the social media platform Reddit, spanning from 2008 to 2016.
- **PeMS04** and **PeMS08** are real-time traffic flow datasets providing traffic information for the state of California, USA. The PeMS04 dataset includes traffic flow data from over 39,000 sensors, while the PeMS08 dataset includes data from over 40,000 sensors. These sensors are located on freeways and arterial roads throughout California. The datasets cover the periods from January 1, 2018, to December 31, 2018, and from January 1, 2020, to December 31, 2020, respectively. Both datasets are collected at 5-minute intervals and include information on traffic speed, occupancy, and volume, resulting in 288 data points per detector per day. Additionally, the datasets include weather information and incident reports, which can be used to analyze the impact of weather and incidents on traffic flow. The data are transformed using zero-mean normalization to ensure the average is 0.
- **Enron**, **FB**, and **COLAB**: These datasets are dynamic graphs constructed from different types of interactions: email messages exchanged between employees, co-author relationships among authors, and Facebook wall posts, respectively. The Enron dataset represents the email communication network of employees at the Enron Corporation, where nodes represent individuals and edges represent email messages sent between them over time. The FB dataset captures the social network of Facebook users, where nodes represent users and edges represent friendship connections. Finally, the COLAB dataset contains transcripts of meetings held by community organizations, where nodes represent participants and edges represent their interactions during the meetings. We collected and processed these three datasets following the methodology described in (Hajiramezanali et al., 2019).

A.3 Baselines

The details about the baselines are as follows:

- (a) **Sensitivity Analysis (SA)** (Baldassarre & Azizpour, 2019) computes importance scores using squared gradients of input features through back-propagation. It assumes that higher absolute gradient values indicate greater importance, but it fails to accurately represent importance and is prone to saturation issues (Shrikumar et al., 2017).
- (b) **GradCAM** (Pope et al., 2019b) extends the CAM (Pope et al., 2019b) method to graph classification by removing the GAP layer constraint and mapping the final node embeddings to the input space for measuring node importance. It uses gradients as weights to combine different feature maps, computed by averaging the gradients of the target prediction with respect to the final node embeddings.
- (c) **GNN-GI** (Schnake et al., 2021) adopts Grad$\odot$Input (GI) (Shrikumar et al., 2017), which quantifies the contribution of features by computing the element-wise product of the input features and the gradients of

the decision function with respect to those features. As a result, GI takes into account both the sensitivity of features and the scale of their values.

- (d) **GNNExplainer** (Ying et al., 2019) generates explanations for predictions in the form of subgraphs and feature masks that highlight the relevant parts of the input data. It provides explanations by generating a compact subgraph from the input graph, along with a select subset of node features that greatly influence the prediction.
- (e) **PGExplainer** (Luo et al., 2020) leverages a deep neural network parameterized explainer to generate global explanations that highlight important subgraphs influencing a model's predictions. This method endows PGExplainer with a natural capacity to deliver multi-instance explanations.
- (f) **SubgraphX** (Yuan et al., 2021) identifies important subgraphs measured by Shapley values. It employs the Monte Carlo tree search algorithm for efficiently exploring various subgraphs within a given input graph.
- (g) **GCN-SE** (Fan et al., 2021) computes the importance of different graph snapshots by measuring the change in accuracy after masking the attention in that timestep.
- (h) **T-GNNExplainer** (Xia et al., 2022) finds a subset of historical events that lead to the prediction, given a temporal prediction of a model. This method regards a temporal graph as a sequence of temporal events between nodes.

A.4 Evaluation Metrics

We present a comprehensive overview of the four key quantitative metrics that have been instrumental in our analysis: confidence, sparsity, stability, and fidelity. The subsequent sections provide a detailed exposition of each metric.

- **Fidelity** characterizes whether the explanations are faithfully important to the model predictions (Sanchez-Lengeling et al., 2020). In the experiment, we measure fidelity by calculating the difference in classification accuracy or regression errors obtained by occluding all nodes with importance values greater than a threshold τ_1 on a scale of $(0, 1)$. We averaged the fidelity across classes for each method.
- **Sparsity** measures the fraction of nodes selected for an explanation (Yuan et al., 2021; Pope et al., 2019b). It evaluates whether the model efficiently marks the most contributive part of the dataset. High sparsity scores indicate that fewer nodes are identified as important. In our experiment, we compute sparsity by calculating the ratio of nodes with saliency values or relevances lower than a predefined threshold τ_2 on a scale of $(0, 1)$.
- **Stability** assesses the consistency of explanations when small changes are applied to the input (Sanchez-Lengeling et al., 2020). Good explanations should be stable, meaning they remain approximately the same under small input perturbations. To evaluate stability, we randomly add more edges at a ratio of $r\%$ and measure the change in relevances/importances produced by the model.

A.5 More Details About Layer-wise Relevance Propagation

In the following, 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

$$x_{k_2} = g\left(\sum_{k_1} w_{k_1 k_2} x_{k_1} + b\right) \tag{22}$$

Given an image x and a classifier $f(\cdot)$ the aim of layer-wise relevance propagation is to assign each pixel p of x a pixel-wise relevance score $R_p^{(1)}$ such that

$$f(x) \approx \sum_p R_p^{(1)} \tag{23}$$

Pixels p with $R_p^{(1)} < 0$ contain evidence against the presence of a class, while $R_p^{(1)} > 0$ is considered as evidence for the presence of a class. These pixel-wise relevance scores can be visualized as an image called a heatmap (see Fig. 1 for examples). Obviously, many possible such decompositions exist which satisfy equation 2. The work of [1] yields pixel-wise decompositions which are consistent with evaluation measures [8] and human intuition.

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 Equation (24) holds.

$$R_{k_2}^{(l+1)} = \sum_{k_1 \in (l)} R_{k_1 \leftarrow k_2}^{(l,l+1)}. \tag{24}$$

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 Equation (25):

$$R_{k_1}^{(l)} = \sum_{k_2 \in (l+1)} R_{k_1 \leftarrow k_2}^{(l,l+1)}, \tag{25}$$

The propagation of relevance from layer $l+1$ to layer l is defined in Equation (24) and Equation (25). 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 $R_d^{(1)}$.

Epsilon Rule (LRP-ϵ) (Bach et al., 2015). A first enhancement of the basic LRP-0 rule consists of adding a small positive term ϵ in the denominator: The work in (Bach et al., 2015) established two formulas for computing the messages $R_{k_1 \leftarrow k_2}^{(l,l+1)}$. The first formula called ϵ-rule is given by

$$R_{k_1 \leftarrow k_2}^{(l,l+1)} = \frac{z_{k_1 k_2}}{z_{k_2} + \epsilon \cdot \operatorname{sign}(z_{k_2})} R_{k_2}^{(l+1)}, \tag{26}$$

with $z_{ij} = (w_{ij} x_i)^p$ and $z_j = \sum_{k: w_{kj} \neq 0} z_{kj}$. The variable ϵ is a "stabilizer" term whose purpose is to avoid numerical degenerations when z_j is close to zero, and which is chosen to be small.

Epsilon Rule (LRP-$\boldsymbol{\epsilon}$) (Bach et al., 2015). A first enhancement of the basic LRP-0 rule consists of adding a small positive term ϵ in the denominator:

$$R_{k_1 \leftarrow k_2}^{(l,l+1)} = \sum_{k_2} \frac{a_{k_1}^{(l)} \mathbf{W}_{k_1 k_2}}{\epsilon + \sum_{k_2, k_1} a_{k_1}^{(l)} \mathbf{W}_{k_1 k_2}} R_{k_2}^{(l+1)}$$

The role of ϵ is to absorb some relevance when the contributions to the activation of neuron k are weak or contradictory. As ϵ becomes larger, only the most salient explanation factors survive the absorption. This typically leads to explanations that are sparser in terms of input features and less noisy.

Therefore, by summing up the relevance over all neurons k_2 in layer $l+1$, based on Equation (26). The Equation (3) can be obtained from Equation (26):

$$R_{k_1 \leftarrow k_2}^{(l,l+1)} = \sum_{k_2} \frac{\mathbf{W}_{k_1 k_2} a_{k_1}^{(l)}}{\epsilon + \sum_{k} \mathbf{W}_{k k_2} a_{k}^{(l)}} R_{k_2}^{(l+1)}.$$

A.6 More Experiments on Dynamic GNN Architectures

We conducted additional experiments on diverse dynamic GNN architectures, including, Evolve-GCN (Pareja et al., 2020), DySAT (Sankar et al., 2018), GC-LSTM (Chen et al., 2022), and ROLAND (You et al., 2022).

Table 3: Experimental results on other dynamic GNNs, in terms of fidelity ($\tau_1 = 0.8$), sparsity ($\tau_2 = 3\times10^{-4}$), and stability ($r = 20\%$).

Dataset	Metric	Evolve-GCN	DySAT	GC-LSTM	ROLAND
Reddit	Fidelity ↑	0.32	0.31	0.27	0.42
	Sparsity ↑	0.88	0.85	0.90	0.81
	Stability ↓	0.14	0.15	0.18	0.21
PeMS04	Fidelity ↑	0.43	0.22	0.36	0.30
	Sparsity ↑	0.99	0.99	0.99	0.99
	Stability ↓	0.11	0.30	0.27	0.23
PeMS08	Fidelity ↑	0.31	0.22	0.21	0.32
	Sparsity ↑	0.95	0.90	0.91	0.90
	Stability ↓	0.16	0.17	0.15	0.11
Enron	Fidelity ↑	0.24	0.20	0.24	0.27
	Sparsity ↑	0.87	0.83	0.82	0.79
	Stability ↓	0.16	0.11	0.19	0.08
FB	Fidelity ↑	0.33	0.37	0.23	0.11
	Sparsity ↑	0.96	0.87	0.94	0.92
	Stability ↓	0.11	0.17	0.19	0.15
COLAB	Fidelity ↑	0.47	0.43	0.41	0.38
	Sparsity ↑	0.98	0.97	0.98	0.99
	Stability ↓	0.11	0.17	0.19	0.15