EXPLAINING TEMPORAL GRAPH MODELS THROUGH AN EXPLORER-NAVIGATOR FRAMEWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

While Graph Neural Network (GNN) explanation has recently received significant attention, existing works are generally designed for static graphs. Due to the prevalence of temporal graphs, many temporal graph models have been proposed, but explaining their predictions still remains to be explored. To bridge the gap, in this paper, we propose a Temporal GNN Explainer (T-GNNExplainer) method. Specifically, we regard a temporal graph as a sequence of temporal events between nodes. Given a temporal prediction of a model, our task is to find a subset of historical events that lead to the prediction. To handle this combinatorial optimization problem, T-GNNExplainer includes an explorer to find the event subsets with Monte Carlo Tree Search (MCTS), and a navigator that learns the correlations between events and helps reduce the search space. In particular, the navigator is trained in advance and then integrated with the explorer to speed up searching and achieve better results. To the best of our knowledge, T-GNNExplainer is the first explainer tailored for temporal graph models. We conduct extensive experiments to evaluate the performance of T-GNNExplainer. Experimental results demonstrate that T-GNNExplainer can achieve superior performance with up to $\sim$50% improvement in Area under Fidelity-Sparsity Curve.

1 INTRODUCTION

Temporal graphs are highly dynamic networks where new nodes and edges can appear at any time. The input is usually regarded as a sequence of events (node i, node j, timestamp t), which means there is an interaction (edge) between node i and j at timestamp t. It is ubiquitous in many real-world applications, such as friendship in social networks (Pereira et al., 2018; Barrat et al., 2021), and user-item interactions in e-commence (Li et al., 2021c). Many applicable temporal graph models (e.g., Jodie (Kumar et al., 2019), TGAT (Xu et al., 2020), TGN (Rossi et al., 2020)) are proposed considering both time dynamics and graph topology. Compared with static GNNs, temporal graph models learn the representation of each node as a function of time and then predict future evolutions, e.g., which interaction will occur and what time node attributes change.

Despite the success, all these models are black boxes and lack transparency. It is opaque how information aggregates and propagates over a graph and how a prediction is affected by historical events. Human-intelligent explanations are critical for understanding the rationale of predictions and providing insights into model characteristics. Explainers could increase the trust and reliability of temporal graph models when they are applied to high-stakes situations, like fraud detection in financial systems (Wang et al., 2021b) and disease progression prediction in healthcare (Li et al., 2021a). Besides, explainers also help check and mitigate the privacy, fairness and safety issues in real-world applications (Doshi-Velez & Kim, 2017).

While currently there are no methods for explaining temporal graph models, some recent explanation methods (e.g., GNNExplainer (Ying et al., 2019), PGExplainer (Luo et al., 2020) and SubgraphX (Yuan et al., 2021)) for static GNNs are the most related. They identify the important nodes, edges and subgraphs for predictions by perturbing the input of GNN models. Obviously, these models cannot be used to explain a well-trained temporal graph model, as they cannot capture the temporal dependency mixed with the graph topology.

Here we propose T-GNNExplainer, an instance-level model-agnostic explainer for temporal graph models. For any prediction of a *target event*, we aim to find out important events from *candidate*

events, which lead to the model's prediction of occurrence (or absence) of it. The candidate events are previously occurred events satisfying spatial and temporal conditions: they are in the k-hop neighborhood based on the message passing mechanism, and their timestamps should be close to that of the target event.

Specifically, T-GNNExplainer takes the advantages of search-based and learning-based GNN explainers together. Generally speaking, a learning-based explainer is inductive to all the target events, and explaining a target event is very quick once trained. A search-based explainer searches for the best result for each target event, which is more specific but time-consuming. While in this work, T-GNNExplainer is designed as a MCTS process with a learned navigator. We pretrain a navigator in advance to learn the inductive relationship between a target event and its candidate events. Then we utilize MCTS to explore the best combination of candidate events given any new target event. The navigator helps to bias the search process, significantly reducing the search time and improving the performance.

We evaluate T-GNNExplainer on both synthetic and real-world datasets for two typical temporal graph models (TGAT and TGN). On synthetic datasets, we simulate temporal events by the multivariate Hawkes process and pre-defined event relation rules. The highly accurate explanations demonstrate that T-GNNExplainer can find an exact influential event set. Since we do not know the ground truth for real-world datasets, the fidelity-sparsity curve is adopted to evaluate the superiority of T-GNNExplainer compared with baselines. We further provide a case study on synthetic datasets to illustrate the practical events found by T-GNNExplainer and navigation information.

2 Related work

2.1 Temporal Graph and Temporal Graph Models

Graphs can be divided into four types by temporal granularity: static graph, graph with time-weighted edges, discrete-time dynamic graph (DTDG) and continuous-time dynamic graph (CTDG) (Kazemi et al., 2020). The typical graph neural networks (e.g., GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GIN (Xu et al., 2018)) can be used for the former two types to learn the static node embeddings. DTDGs are sequences of static graph snapshots taken at intervals in time. CTDGs are more general and are represented as a sequence of timestamped events, including edge/node addition, deletion, and feature transformations. In this work, we consider temporal graphs as CTDGs and take a sequence of timestamped events as model input since CTDGs are mainstream dynamic graphs with the finest time granularity.

Instead of static node embeddings, temporal graph models are required to learn dynamic node embeddings. DeepCoevelve (Dai et al., 2016) used RNNs to update node embeddings when some nodes are involved in new events. Jodie (Kumar et al., 2019) added the time projection module to make node embeddings evolve over time. However, they lack a GNN-like aggregation from node neighbors, which leads to the staleness problem (i.e., some node embeddings are out of date (Rossi et al., 2020; Kazemi et al., 2020)). Thus, CoPE (Zhang et al., 2021) and TGAT (Xu et al., 2020) are proposed to utilize the message passing mechanism to update node embeddings by its own events and its neighbors' events. It has been demonstrated to improve expressive power. TGN (Rossi et al., 2020) is an up-to-date framework and claims that most previous models are its specific cases. We choose the state-of-the-art TGAT and TGN as target models to be explained in the paper.

2.2 Graph Explainers

One popular way of explaining static graphs is to study the output variations of well-trained GNN models with respect to different input perturbations (Yuan et al., 2020b). Intuitively, the output changes vastly when critical nodes, edges, or subgraphs are perturbed. There are mainly two approaches assigning importance scores to graph entities by perturbations: learning-based and search-based. Learning-based methods (Luo et al., 2020; Shan et al., 2021; Vu & Thai, 2020) leverage node representations generated by the trained GNN and adopt a neural network to learn crucial nodes/edges. They are trained with multiple instances, i.e., learning inductive explanation characteristics for multiple ones. Once trained, inference on new ones is fast. Besides, search-based methods (Yuan et al. (2021); Wang et al. (2021a)) utilize heuristic search algorithms with a score function (e.g., defined by Shapley value or causality) to find an important input subset. Their inference time

per instance is longer because the search space of each instance is different, and they need to explore feasible solutions one by one. There are also some works possessing intrinsically interpretable architectures (Han et al. (2020); Li et al. (2021b); Xiao et al. (2022)) or generating model-level explanations (Yuan et al. (2020a); Shin et al. (2022)). Most of the self-interpretable models seek to a sparse subgraph during forward computation supported by the attention mechanism (Han et al. (2020)) or other internal scores (Li et al. (2021b); Cui et al. (2021)). There is a concurrent work that also explains temporal graph models He et al. (2022). However, they use discrete snapshots of a temporal graph while we focus on continuous event streams. Besides, model-level explainers generally output several static graph prototypes as interpretations. However, such prototypes may not exist to cover complicated interaction dynamics on temporal graphs and ignore local influences on specific events. In this work, we mainly focus on instance-level post-hoc explanation methods since most of the existing state-of-the-art temporal graph models in the literature are not designed with specific consideration on explanations.

3 PRELIMINARY

3.1 TEMPORAL GRAPH MODEL

Assume that the input of temporal graph models is a sequence of events $\mathcal{S} = \{e_1, e_2, \cdots\}$. Each $e_i = \{n_{u_i}, n_{v_i}, t_i, \text{att}_i\}$ means that the node n_{u_i} and n_{v_i} have an interaction (edge) at timestamp t_i with edge attribute att_i. The att_i could be the interaction feature, or an indicator to represent e_i is edge addition/deletion. Further, e_i could involve only one node, $\{n_{u_i}, \text{null}, t_i, \text{att}_i\}$, to represent a node-wise event (node addition/deletion, or node attribute change). These events $\mathcal{S}$ constitute a temporal graph $\mathcal{G} = (\mathcal{N}, \mathcal{S})$ where $\mathcal{S}$ can be regarded as timestamped edges and $\mathcal{N}$ are nodes involved in $\mathcal{S}$. Since $\mathcal{G}$ and $\mathcal{S}$ are mutually defined, we regard $\mathcal{G}$ as both a temporal graph and a set of events in the following.

We utilize the setting defined in (Kazemi et al., 2020) to unify different temporal graph models as an encoder-decoder framework. The encoder is to learn the dynamic embedding for each node over time, and the decoder utilizes the node embeddings for downstream prediction tasks, such as future edge prediction. Specifically, let $\mathcal{G}^i$ denote the graph constructed just before the timestamp t_i, i.e., containing the events $\{e_1, \cdots, e_{i-1}\}$ but excluding e_i. The encoder takes $\mathcal{G}^i$ as the input and obtains Z^i where $Z^i_{n_*}$ is the current embedding of the node n_* at timestamp t_i. The decoder constructs the loss by predicting whether an interaction between a pair of nodes happen at this timestamp (the positive sample is $e_i = \{n_{u_i}, n_{v_i}\}$ while negative samples are the remaining pairs). Thus, the decoder uses Z^i to predict the logit/probability of a new event between any pair of nodes, computes the loss to backpropagate the gradients, and updates the model parameters.

$$\textbf{Encoder}(\mathcal{G}^i) \rightarrow Z^i \qquad \textbf{Decoder}(Z^i) \rightarrow \text{Logit/Probablity of Events} \rightarrow \text{Loss}$$

Let $f(\cdot)$ denote a well-trained temporal graph model including the encoder and the decoder to simplify the notation. $f(\mathcal{G}^i)[e_j]$ means that we use the encoder to compute Z^i by $\mathcal{G}^i$ at timestamp t_i and computes the logit/probability of event e_j by leveraging $Z^i_{n_{u_j}}$ and $Z^i_{n_{v_j}}$.

3.2 PROBLEM FORMULATION

Given the sequence of events and a well-trained temporal graph model $f(\cdot)$, the temporal explainer explains why the model predicts an event e_k would occur or not. Specifically, we aim to find out a subset of events $\mathcal{R}^k$ from all the previous events $\mathcal{G}^k$ to maximize the mutual information $MI(Y_k, \mathcal{R}^k)$. Y_k is the original prediction decided by $f(\mathcal{G}^k)[e_k]$, which is 0 or 1. $\mathcal{R}^k$ determines the distribution of the new prediction by $f(\mathcal{R}^k)[e_k]$. When they are strongly dependent, $\mathcal{R}^k$ is regarded as a good explanation for the prediction of occurrence/absence of event e_k.

According to (Ying et al., 2019), maximizing the mutual information $MI(Y_k, \mathcal{R}^k)$ is equivalent to minimizing conditional entropy $H(Y_k|\mathcal{R}^k)$ because $H(Y_k)$ is a constant. Then we can transform $H(Y_k|\mathcal{R}^k)$ into a cross entropy loss based on (Farnia & Tse, 2016):

$$\min_{\mathcal{R}^k} - \sum_{c=0,1} \mathbb{1}(Y_k = c) \log P(Y_{\text{new}} = c|\mathcal{R}^k) \tag{1}$$

$P(Y_{\text{new}}|\mathcal{R}^k)$ is calculated by $f(\mathcal{R}^k)[e_k]$ and c is the label indicating whether the event occurs or not.

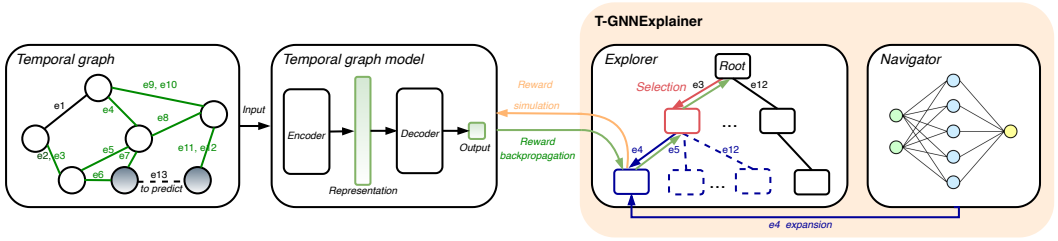


Figure 1: The framework of T-GNNExplainer. We first pre-train the navigator to learn the inductive relationship between events. Then we invoke the explorer to search out a specific combination of important events based on MCTS, including node selection, node expansion, reward simulation and backpropagation. When expanding a node, the navigator infers to decide which event is removed.

Objective. Given a target event e_k on which a prediction is made by $f(\cdot)$, T-GNNExplainer is a mapping g to infer the important events $\mathcal{R}^k = g(e_k, \mathcal{G}^k, f(\cdot))$ by current temporal graph $\mathcal{G}^k$. T-GNNExplainer g can also minimize the cross entropy loss for K target events. Formally, the optimal explainer g^* is defined as

$$g^* = \arg\min_g -\frac{1}{K}\sum_{k=1}^{K}\left[\mathbb{1}(Y_k = 1)\log\sigma(f(\mathcal{R}^k)[e_k]) + \mathbb{1}(Y_k = 0)\log(1-\sigma(f(\mathcal{R}^k)[e_k]))\right] \quad (2)$$
$$\text{subject to} \quad \mathcal{R}^k = g(e_k, \mathcal{G}^k, f(\cdot)) \text{ and } \mathcal{R}^k \subseteq \mathcal{G}^k \text{ and } |\mathcal{R}^k| \leq N_r$$

Here we assume $f(\mathcal{R}^k)[e_k]$ is a single-dimensional logit value produced by the model $f(\cdot)$, and $\sigma(\cdot)$ indicates the sigmoid function. $\mathcal{R}^k$ should be concise so we use N_r as a hyper-parameter to control the size of $\mathcal{R}^k$.

4 Methodology

4.1 Overview

Here we introduce how to identify an important subset of events $\mathcal{R}^k$ for the target event e_k. It is a combinatorial optimization problem where any subset of $\mathcal{G}^k$ whose size is equal to or smaller than N_r could be the explanation. Besides, the search space of a temporal graph explainer is more significant than that of static graph explainers because of duplicate timestamped edges and node-wise events. Therefore, we design an explorer-navigator framework to effectively and efficiently obtain $\mathcal{R}^k$, shown in Fig. 1. The navigator is trained from multiple target events to capture the inductive correlation between events. The explorer is guided by the navigator and finds a more specific result based on Monte Carlo Tree Search. We will present them in the following.

4.2 Navigator

Inspired by previous parameterized explainers (Luo et al., 2020), we pretrain a navigator to provide a global understanding of the relationship among events. Concretely, the navigator is a feed-forward neural network $h_\theta(e_j, e_k)$, which infers the importance score of an event e_j w.r.t a target event e_k. The scores will be leveraged in the node expansion of the explorer (Sec. 4.3) to facilitate the searching. The input features and the training/inference process are described as follows.

Navigator features. To capture the correlation between the target event $e_k = \{n_{u_k}, n_{v_k}, t_k, \text{att}_k\}$ and each candidate event $e_j = \{n_{u_j}, n_{v_j}, t_j, \text{att}_j\}$, we construct navigator input features as:

$$Z_{e_k,e_j} = [X_{n_{u_k}} || X_{n_{v_k}} || \text{ Time}(t_k) \text{ } || \text{ att}_k \text{ } || X_{n_{u_j}} || X_{n_{v_j}} || \text{ Time}(t_j) \text{ } || \text{ att}_j \text{ }]^T \quad (3)$$

X represents the node feature matrix. $\text{Time}(\cdot)$ is a function converting a real-valued timestamp to a vector that could be learnable (Xu et al., 2020; Rossi et al., 2020) or not (Vaswani et al., 2017). In this paper, we adopt the harmonic encoder (Xu et al., 2020) as the time function. We input all the candidate events w.r.t a target event as a batch into h_θ.

Training Process. We use the same objective function described in Eq. 2. The output of h_θ is regarded as a soft-mask assigning weights to corresponding temporal edges of $\mathcal{G}^k$. If the model $f(\cdot)$ is differentiable w.r.t edge weights, we add the soft-mask as edge aggregation weights in the model, and then obtain the new prediction. Otherwise, the reparameterization trick is used in $f(\cdot)$. More details about reparameterization can be found in (Luo et al., 2020). Note that h_θ is trained inductively, i.e., events in the training set are different from those to be explained by T-GNNExplainer.

Inference Process. Provided with a candidate event e_j and a target event e_k, we construct the input Z_{e_k,e_j}, put it into the trained navigator h_θ and infer the score, which will be utilized by the explorer.

4.3 EXPLORER

We adopt Monte Carlo Tree Search (MCTS) in the explorer. First of all, we initialize the root node as a set of candidate events. Then multiple rounds (a.k.a., rollouts) are conducted to expand nodes in the search tree, where each node represents a feasible subset of events in the search space. There are four aspects in each round: (1) select a path from the root to a leaf node; (2) expand new children by removing unimportant events according to the navigator in the path selecting procedure; (3) simulate reward of new nodes by temporal graph model; (4) backpropagate the leaf node's reward to update information in path nodes. At last, a node achieving the best reward and satisfying the sparsity threshold is our final explanation result. We present the pseudo-code in Appendix.

4.3.1 INITIALIZATION

The root node includes all the candidate events, which are previously occurred events satisfying spatial and temporal conditions. Take the temporal graph in Fig. 1 as an example where we explain e_{13}. Assume that the encoder uses the 2-hop GNN-based aggregation, the set of seen events by encoder is $\{e_2, \cdots, e_{12}\}$. A temporal threshold is used to remove old events. If the threshold is set as 10, we preserve 10 recently occurred events. Finally, the root is initialized as $\{e_3, \cdots, e_{12}\}$.

4.3.2 NODE SELECTION

We use $\mathcal{N}^i$ to represent a node in the search tree and use e_j to indicate one action, i.e., discard the event e_j from $\mathcal{N}^i$. We follow the UCT (Upper Confidence bound applied to Trees) formula proposed in (Kocsis & Szepesvári, 2006) to balance the exploitation and the exploration in the node selection. Assume we are on node $\mathcal{N}^i$, the action criteria is

$$e^* = \underset{e_j \in \mathcal{C}(\mathcal{N}^i)}{\arg\max} \left(\frac{c(\mathcal{N}^i, e_j)}{n(\mathcal{N}^i, e_j)} + \lambda \frac{\sqrt{\sum_{e_l \in \mathcal{C}(\mathcal{N}^i)} n(\mathcal{N}^i, e_l)}}{1 + n(\mathcal{N}^i, e_j)} \right) \tag{4}$$

$\mathcal{C}(\mathcal{N}^i)$ indicates the events already expanded in $\mathcal{N}^i$, $n(\mathcal{N}^i, e_j)$ is the count for selecting e_j on node $\mathcal{N}^i$ in previous rollouts, and $c(\mathcal{N}^i, e_j)$ denotes the cumulative reward of selecting e_j on node $\mathcal{N}^i$. The first component is exploitation: we select the node with a high average reward. The second component corresponds to exploration: we select the node with few simulations. We select and move to $\mathcal{N}^i$'s child node by removing the event e^* from $\mathcal{N}^i$.

4.3.3 NODE EXPANSION

The strategy of node expansion significantly influences the performance because it affects the search space and hence the best node's quality. Previous works expand all possible children for any selected node (Yuan et al., 2021). Instead, we only expand the best potential node to refine the search space. Assuming the selected node $\mathcal{N}^i$ is expandable (i.e., the number of children is less than the number of events contained in the node), the explorer invokes the navigator (Sec. 4.2) to obtain potential scores:

$$e^* = \underset{e_j \in \mathcal{N}^i / \mathcal{C}(\mathcal{N}^i)}{\arg\min} h_\theta(e_j, e_k) \tag{5}$$

$\mathcal{N}^i / \mathcal{C}(\mathcal{N}^i)$ means possible events which are not expanded in the previous rollouts. We remove the most unimportant event e^* to expand a new node. Since the navigator is learned in advance and infers the score quickly, the additional cost is negligible. We could also expand the top-k candidates or

induce randomness to trade off the exploitation and exploration in the expansion step. For example, we select e^* to expand with probability $1-\epsilon$ and select a random unexplored e_j with probability ϵ.

The node selection and expansion are done alternatively. We start from the root node. We expand new child node(s) for the root according to Eq. 5. Then we choose the node with the highest value based on Eq. 4 from the root's original and new child nodes and move to it. Next we repeat the expansion and selection from the new node. The process ends when the current node is identified as a leaf node, e.g., the node has less than five events.

4.3.4 Reward Simulation and Backpropagation

The reward is simulated by the temporal graph model. In detail, we compute the reward of a leaf node $r(\mathcal{N}^{\text{leaf}})$ by computing the *negative* cross entropy loss [1] using Eq. 1, where $\mathcal{R}^k = \mathcal{N}^{\text{leaf}}$. In backpropagation, all the nodes $\mathcal{N}^i$ from root to the leaf will update $n(\cdot,\cdot)$ and $c(\cdot,\cdot)$ by adding the leaf node's reward and one respectively, i.e., $c(\mathcal{N}^i, e_l) = c(\mathcal{N}^i, e_l) + r(\mathcal{N}^{\text{leaf}})$, $n(\mathcal{N}^i, e_l) = n(\mathcal{N}^i, e_l) + 1$. e_l is the action selected at $\mathcal{N}^i$.

5 Experiments

In this section, we evaluate the performance of T-GNNExplainer with several baseline explainers. We first describe synthetic datasets, real-world datasets and target models in Sec. 5.1. Then we present the detailed experimental setup, including baselines and evaluation metrics in Sec. 5.2. Sec. 5.3 is a quantitative evaluation to demonstrate that T-GNNExplainer could surpass the baselines up to $\sim$50% improvement in the Area Under the Fidelity-Sparsity Curve (AUFSC). Furthermore, we investigate the navigator's effect in Sec. 5.4. Finally, a case study in Sec. 5.5 is constructed to show the explanations provided by T-GNNExplainer and navigation weights. The dataset statistics, target models' performance, and running time of all the methods are presented in Appendix. The code and datasets are attached in the supplementary.

5.1 Datasets and target models

Real-world datasets: We adopt two typical real-world temporal graphs: Wikipedia[2] and Reddit[3]. The Wikipedia dataset consists of $\sim$9300 active users and top edited pages and $\sim$160,000 temporal edges. A 172-dimensional user editing feature accompanies each temporal edge. The Reddit dataset is analogous to Wikipedia with $\sim$11,000 active users and subreddits and $\sim$700,000 temporal edges. The 172-dimensional temporal edge features come from user post contents.

Synthetic datasets: We utilize the Hawkes process (Hawkes, 1971) and tick library (Bacry et al., 2017) to generate synthetic datasets. In Fig. 2, we define four types of events ($E_1 \sim E_4$) in a graph. According to the multivariate Hawkes process, the intensity of an event is divided into two parts: endogenous and exogenous. For example, E_0 has endogenous intensity 0.5 to happen because of itself, and E_3 has exogenous intensity 2 influenced by the happening of E_2. Given a pre-defined event relation including endogenous/exogenous intensities, we adopt the tick library to simulate a sequence of events with timestamps. We generate two synthetic datasets with $\sim$ 10000 timestamps based on event relation v1 and v2 in Fig. 2. More details are described in Appendix.

Target models: We adopt two recent state-of-the-art temporal graph models TGAT (Xu et al., 2020) and TGN (Rossi et al., 2020). TGAT presents a temporal attention layer to aggregate a node's previous neighbours in chronological order and a time encoder to encode temporal information. TGN further proposes a memory store and update module to persist each node's temporal state. A point process model Transformer Hawkes Process (THP) is also compared in Sec. A.8 in Appendix. These models are trained in a self-supervised manner for real-world datasets, i.e., events seen on the graph are positive samples and randomly chosen unseen events are negative ones. For synthetic datasets, happened E_3 timestamps are positive samples, and we uniformly sample random timestamps as negative samples.

[1] Because we use a larger reward to indicate a better solution in MCTS, a negative CR is consistent with our overall objective in Eq. 2.

[2] http://snap.stanford.edu/jodie/wikipedia.csv

[3] http://snap.stanford.edu/jodie/reddit.csv

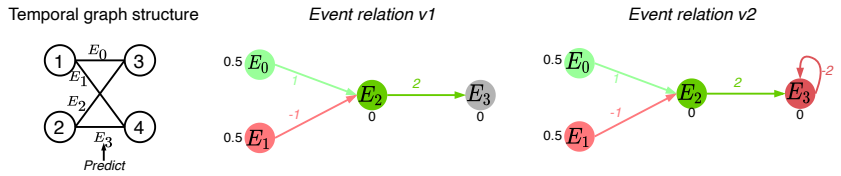


Figure 2: Synthetic temporal graph and pre-defined event relations. Values on nodes indicate endogenous intensities. Values on edges define exogenous intensities. Green edges are positive influences while red edges are negative influences. Grey edges reflect no influence.

Table 1: Best fidelity (↑) and AUFSC (↑) achieved by each explainer on real-world datasets.

	Wikipedia				Reddit			
	TGAT		TGN		TGAT		TGN	
	Best Fid	AUFSC	Best Fid	AUFSC	Best Fid	AUFSC	Best Fid	AUFSC
ATTN	0.891	0.564	<u>0.479</u>	<u>0.073</u>	0.658	-0.654	0.575	<u>0.289</u>
PBONE	0.027	-2.227	0.296	-0.601	0.167	-2.492	0.340	-0.256
PG	<u>1.354</u>	<u>0.692</u>	0.464	-0.231	<u>0.804</u>	<u>-0.369</u>	<u>0.679</u>	0.020
T-GNNExplainer	**1.836**	**1.477**	**0.866**	**0.590**	**1.518**	**1.076**	**1.362**	**1.113**

5.2 Baselines and Setups

Baselines: We compare the performance of T-GNNExplainer with several baseline methods. (1) We implement PGExplainer (PG) in (Luo et al., 2020) and adapt it for the temporal graph scenario. The adapted PG computes a weight to each event instead of each edge. The input information for event e_j is the same as the Z_{e_i,e_j} in Eq. 3. We add the output score of PG to the attention weights in the target model for all layers and use the same training objectives as T-GNNExplainer. (2) For the attention-based explainer (ATTN), we extract the attention weights in TGAT/TGN and average the values over all layers. The averaged weights are regarded as importance scores. (3) Besides, we implement a straightforward explainer by perturbing one candidate event (PBONE), i.e., we compute the importance of each event $e_j \in \mathcal{G}^k$ by feeding $\mathcal{G}^k/\{e_j\}$ into the target model. Moreover, we also compare with the self-interpretable THP-based THPExplainer in Sec. A.8 in Appendix.

Evaluation metrics: We adopt the fidelity $Fid(f(\mathcal{G}^k)[e_k], f(\mathcal{R}^k)[e_k])$ and the sparsity $Sp(\mathcal{R}^k, \mathcal{G}^k)$ to evaluate the performance. Instead of using the difference between the original and new prediction probabilities to define the fidelity in the previous work (Yuan et al., 2020b), we use the difference between logits because logits could exhibit explainers' performance more clearly. The fidelity is defined as $Fid(f(\mathcal{G}^k)[e_k], f(\mathcal{R}^k)[e_k]) = \mathbb{1}(Y_k = 1)(f(\mathcal{R}^k)[e_k] - f(\mathcal{G}^k)[e_k]) + \mathbb{1}(Y_k = 0)(f(\mathcal{G}^k)[e_k] - f(\mathcal{R}^k)[e_k])$. Besides, sparsity is defined as $Sp = |\mathcal{R}^k|/|\mathcal{G}^k|$. The higher fidelity and higher sparsity mean a better result. We draw the fidelity-sparsity curve and compute area under the curve **AUFSC** to evaluate the performance. A larger AUFSC indicates a better performance. Note that AUFSC may be negative because fidelity could be negative.

Furthermore, we also use the metric **Best Fid**, indicating the best fidelity ever found by the explainer without sparsity limitations. For T-GNNExplainer, we traverse all tree nodes to find a node with the best fidelity. For baseline explainers, we rank all the candidate events in ascending order by their importance scores produced by the explainer and successively preserve top events to find the subset with the best fidelity.

Experimental setup: We use a two-layer MLP with 128 hidden units to instantiate the navigator h_θ. We set the exploration parameter λ to 5 and the rollout number to 500 in the explorer. Following the same setting in TGAT and TGN, we adopt a two-layer attention architecture and harmonic encoding for timestamps. We train both TGAT and TGN with a 70%, 15%, and 15% splitting scheme of datasets based on timestamps. For all methods, we limit the number of candidate events to 25 and randomly sample 500 events in the test dataset as target events for the explanation. We use a machine with an RTX 2080 GPU and a 48-core Intel(R) Xeon(R) CPU@2.2GHz. More hyper-parameters are listed in Appendix.

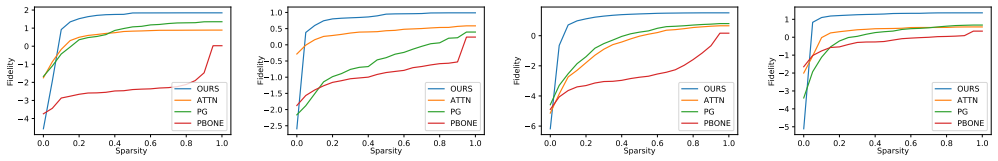
Table 2: Best fidelity (↑) and AUFSC (↑) achieved by each explainer on synthetic datasets.

	Synthetic v1				Synthetic v2			
	TGAT		TGN		TGAT		TGN	
	Best Fid	AUFSC	Best Fid	AUFSC	Best Fid	AUFSC	Best Fid	AUFSC
ATTN	<u>0.555</u>	<u>0.390</u>	<u>2.178</u>	<u>1.624</u>	0.605	<u>0.291</u>	0.988	<u>-0.634</u>
PBONE	0.044	-2.882	0.000	-3.311	0.096	-4.771	0.320	-5.413
PG	0.476	-0.081	2.006	0.626	<u>1.329</u>	-0.926	<u>1.012</u>	-1.338
T-GNNExplainer	**0.780**	**0.666**	**2.708**	**2.281**	**1.630**	**1.331**	**4.356**	**3.224**

5.3 Performance comparison with baselines

In this section, we report the quantitative results in Table 1 and Table 2 for real-world and synthetic datasets respectively[4]. We find that T-GNNExplainer outperforms baseline explainers significantly and consistently for two metrics on all the datasets. On the real-world datasets, the gains of AUFSC (Best Fid) are up to 53%(26%), 86%(45%), 134%(47%), and 74%(50%) w.r.t to the leading baseline in four scenarios. ATTN and PG obtain comparable performance while PBONE performs the worst. The results of synthetic datasets are analogous to those of real-world datasets.

Besides, we illustrate the fidelity-sparsity curve on the real-world datasets intuitively, shown in Fig. 3. T-GNNExplainer achieves the highest final fidelity than other baselines and is also the highest one under a given sparsity threshold. Moreover, with a relatively small sparsity threshold, e.g., 0.2, T-GNNExplainer can already find a solution with a high fidelity compared to its final best value, which indicates that T-GNNExplainer explores the low-sparsity event subsets efficiently. Without the searching procedure, the fidelity of PG and ATTN increase slowly. PBONE performs the worst in all scenarios because it treats each event independently. More performance investigations of the navigator are illustrated in Sec. A.5 and Sec. A.6 in Appendix.



(a) TGAT, Wikipedia (b) TGN, Wikipedia (c) TGAT, Reddit (d) TGN, Reddit

Figure 3: Fidelity-sparsity comparison of explainers on real-world datasets with target models.

5.4 Efficiency influence of the navigator

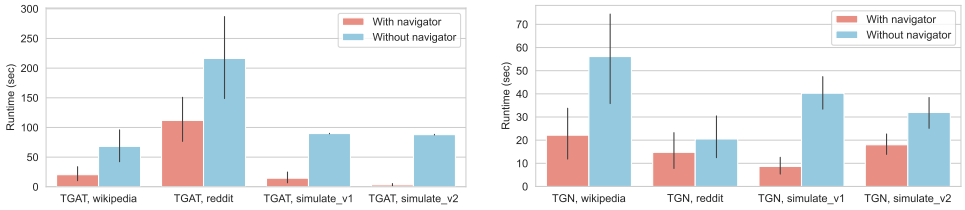
In this section, we investigate the efficiency enhancement of the navigator. We set the rollout number to 500 for *with navigator* and *without navigator*. Here, we compare the time exhausted to achieve a fidelity threshold. Let the *best fidelity* denote the larger final fidelity between *with navigator* and *without navigator*. We set the fidelity threshold to $0.8\times$*best fidelity* to compare the time exhausted.

Results for all the datasets and both models are shown in Fig. 4. The results are averaged over all target events. We can find that the running time of *with navigator* is always less than that of *without navigator* under all settings. The efficiency improvement of *with navigator* is about 70.1%, 60.1%, 48.2%, and 28.1%. On the synthetic datasets, we can observe similar results as those on real-world datasets. The speedup of the navigator is about 83.85%, 96.43%, 78.48%, and 43.66% respectively under four synthetic settings. Overall, the navigator effectively speeds up the searching procedure of the explorer to achieve reasonable solutions. More runtime comparisons with baselines and complexity analysis of T-GNNExplainer are illustrated in Sec. A.4 in Appendix.

5.5 Case study

In this section, we visualize explanations for target events with TGAT on the synthetic dataset. Specifically, we show the scores given by T-GNNExplainer, the navigator, and the target models'

[4]Best results are in bold and the second best are underlined.

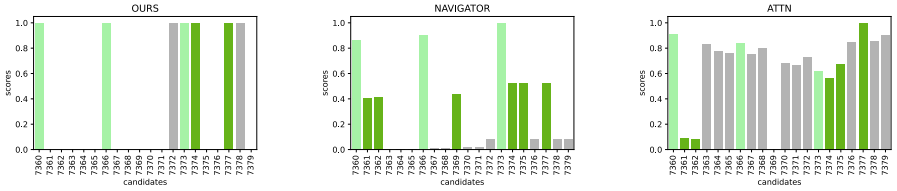


(a) TGAT (b) TGN

Figure 4: Efficiency of the navigator on all the datasets with target models.

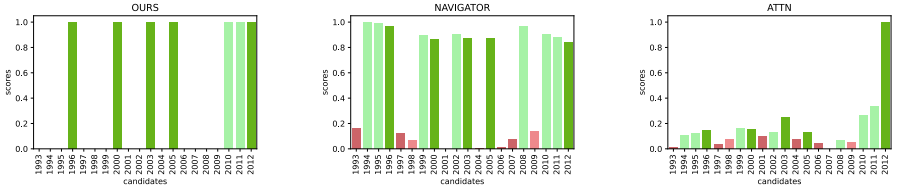
intrinsic attention module. The values are normalized in [0, 1]. Since T-GNNExplainer returns a subset, we assign 1 to the selected events and 0 to others. The bar colours are consistent with type colours in Fig. 2. The light green and green colours represent positive events to trigger the event E_3, while the red colour indicate negative events to inhibit the event E_3. The grey are irrelevant events. Because we explain the happened target event E_3, a good explainer should assign relatively high scores to green bars and low scores to gray and red events.

Fig. 5 and Fig. 6 present the cases on synthetic v1 and v2. ATTN finds a dense event subset including irrelevant or negative events (Fig. 5(c)), or overlooks some previous positive events (Fig. 6(c)). The navigator obtains a better result than ATTN. It assigns high scores to green bars and almost eliminates the gray and red ones. Assisted by the navigator, T-GNNExplainer utilizes the explorer to further filter the events and make sure the final result is concise. Overall, our final event set is sparse while it remains the most important events leading to the occurrence of the target event E_3.



(a) OURS (b) NAVIGATOR (c) ATTN

Figure 5: Synthetic v1, target event index 7380.



(a) OURS (b) NAVIGATOR (c) ATTN

Figure 6: Synthetic v2, target event index 2013.

6 Conclusion

In this paper, we propose T-GNNExplainer which is the first explainer for temporal graph models. We design a novel explorer-navigator framework to search for the explanation of temporal predictions effectively and efficiently. Experimental results show the superiority of T-GNNExplainer on both synthetic and real-world datasets with two typical temporal graph models.

In the future, we plan to explore model-level explainers for temporal graphs because the instance-level explanation is local and may be sensitive to the specific input instance. Besides, we generate two synthetic datasets using the Hawkes process but practical networks are more complex and include rich patterns. How to generate simulated datasets using other generation algorithms to mimic the real-world dynamic graphs deserves further investigation.

REFERENCES

Emmanuel Bacry, Martin Bompaire, Stéphane Gaïffas, and Soren Poulsen. Tick: a python library for statistical learning, with a particular emphasis on time-dependent modelling. *arXiv*, 2017.

Alain Barrat, Valeria Gelardi, Didier Le Bail, and Nicolas Claidiere. From temporal network data to the dynamics of social relationships. *Proc. Royal Soc. B*, 288(20211164), 2021.

Hejie Cui, Wei Dai, Yanqiao Zhu, Xiaoxiao Li, Lifang He, and Carl Yang. Brainnnexplainer: An interpretable graph neural network framework for brain network based disease analysis. *arXiv preprint arXiv:2107.05097*, 2021.

Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv*, 2016.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv*, 2017.

Farzan Farnia and David Tse. A minimax approach to supervised learning. *Advances in Neural Information Processing Systems*, 29, 2016.

Zhen Han, Peng Chen, Yunpu Ma, and Volker Tresp. Explainable subgraph reasoning for forecasting on temporal knowledge graphs. In *International Conference on Learning Representations*, 2020.

Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

Wenchong He, Minh N Vu, Zhe Jiang, and My T Thai. An explainer for temporal graph neural networks. *arXiv preprint arXiv:2209.00807*, 2022.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *JMLR*, 21(70):1–73, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, pp. 282–293. Springer, 2006.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*, pp. 1269–1278, 2019.

Shuang Li, Mingquan Feng, Lu Wang, Abdelmajid Essofi, Yufeng Cao, Junchi Yan, and Le Song. Explaining point processes by learning interpretable temporal logic rules. In *ICLR*, 2021a.

Xiaoxiao Li, Yuan Zhou, Nicha Dvornek, Muhan Zhang, Siyuan Gao, Juntang Zhuang, Dustin Scheinost, Lawrence H Staib, Pamela Ventola, and James S Duncan. Braingnn: Interpretable brain graph neural network for fmri analysis. *Medical Image Analysis*, 74:102233, 2021b.

Zhao Li, Pengrui Hui, Peng Zhang, Jiaming Huang, Biao Wang, Ling Tian, Ji Zhang, Jianliang Gao, and Xing Tang. What happens behind the scene? towards fraud community detection in e-commerce from online to offline. In *WWW*, 2021c.

Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. In *NeurIPS*, volume 33, pp. 19620–19631, 2020.

Fabíola SF Pereira, João Gama, Sandra de Amo, and Gina Oliveira. On analyzing user preference dynamics with temporal social networks. *Machine Learning*, 107(11):1745–1773, 2018.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML*, 2020.

Caihua Shan, Yifei Shen, Yao Zhang, Xiang Li, and Dongsheng Li. Reinforcement learning enhanced explainer for graph neural networks. In *NeurIPS*, volume 34, 2021.

Yong-Min Shin, Sun-Woo Kim, Eun-Bi Yoon, and Won-Yong Shin. Prototype-based explanations for graph neural networks (student abstract). 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, volume 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

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

Xiang Wang, Yingxin Wu, An Zhang, Xiangnan He, and Tat seng Chua. Causal screening to interpret graph neural networks, 2021a. URL `https://openreview.net/forum?id=nzKv5vxZfge`.

Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *SIGMOD*, pp. 2628–2638, 2021b.

Tingsong Xiao, Lu Zeng, Xiaoshuang Shi, Xiaofeng Zhu, and Guorong Wu. Dual-graph learning convolutional networks for interpretable alzheimer's disease diagnosis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 406–415. Springer, 2022.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *ICLR*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2018.

Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In *NeurIPS*, pp. 9240–9251, 2019.

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

Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *arXiv*, 2020b.

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

Yao Zhang, Yun Xiong, Dongsheng Li, Caihua Shan, Kan Ren, and Yangyong Zhu. Cope: Modeling continuous propagation and evolution on interaction graph. In *CIKM*, pp. 2627–2636, 2021.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

A APPENDIX

A.1 PSEUDOCODE

Algorithm 1: The T-GNNExplainer algorithm.

Input : target model f, temporal graph $\mathcal{G}^k$, target event e_k, sparsity q, navigator h_θ, rollout N.
Output: the best subset $\mathcal{N}^*$ satisfying sparsity threshold q.

Initialize tree $\mathcal{T} = \{\mathcal{G}^k\}$;
for $i = 1, \cdots, N$ **do**
  $CurrNode$ = $\mathcal{G}^k$;
  $PathNodes$ = [$CurrNode$];
  /* move downwardly if the current node is not a leaf. */
  while $|CurrNode|/|\mathcal{G}^k| > q$ **do**
    if $CurrNode$ *is expandable* **then**
      /* expand one child using the node expansion strategy with the help of the navigator. */
      $Child$ = `NodeExpansionStrategy`($CurrNode$);
      $CurrNode.children$.append($Child$);
      $\mathcal{T}$.append($Child$);
    /* select one child using the node selection strategy. */
    $CurrNode$ = `NodeSelectionStrategy`($CurrNode.children$);
    $PathNodes$.append($CurrNode$);
  end
  $LeafNode$=$CurrNode$;
  $LeafNode.r$ = `RewardFunction`($LeafNode, e_k$);
  /* update path nodes' statistics using the reward. */
  `UpdateStatistics`($PathNodes$, $LeafNode.r$);
end
/* find the best node having the largest reward and satisfying the sparsity criteria $|Node|/|\mathcal{G}^k| \leq q$. */
$\mathcal{N}^*$ = `FindBestTreeNode`($\mathcal{T}$, q);
return $\mathcal{N}^*$;

A.2 SYNTHETIC DATASET

Hawkes process: The multivariate Hawkes process (MHP) is the counting process where an arrival of an event can affect the arrival rates of other events. In our synthetic datasets, we use a MHP to capture mutual excitation/inhibition and generate a sequence of events with timestamps.

We assume that there are D event types where the type of an event (node n_u, node n_v, timestamp t) is decided by its nodes n_u and n_v. The intensity function of the i-th type of events at time t is

$$\lambda_i(t) = \mu_i + \sum_{j=1}^{D} \int_0^t \varphi_{ij}(t-s) \mathrm{d}N_j(s) \tag{6}$$

where $\varphi_{ij}(x) = A_{ij}\theta e^{-\theta x}$.

The first term μ_i indicates the endogenous intensity of event type i, while the second term φ_{ij} represents the exogenous influence from other events. In particular, $N_j(s)$ counts the number of occurred event type j within $[0, s]$, and one arrival of event type j at time s will affect the intensity of event type i at time t by the amount $\varphi_{ij}(t-s)$ for $t > s$. A_{ij} is the influence matrix and θ is a time decay factor. Obviously, we could pre-define the parameters μ_i, A_{ij} and θ to decide a MHP.

We show our parameter setting in Fig. 2, where we have 4 types of events $E_0 - E_3$. For two synthetic datasets, we set $\mu_0 = \mu_1 = 0.5$ and $\mu_2 = \mu_3 = 0$. In Synthetic v1, we set $A_{20} = 1$, $A_{21} = -1$, and $A_{32} = 1$. Other A_{ij} are 0. Therefore, E_0 and E_2 work together to let E_3 happen, while E_1 inhibits

E_3. In Synthetic v2, we set $A_{33} = -2$ on the basis of Synthetic v1. The occurrence of E_3 inhibits itself.

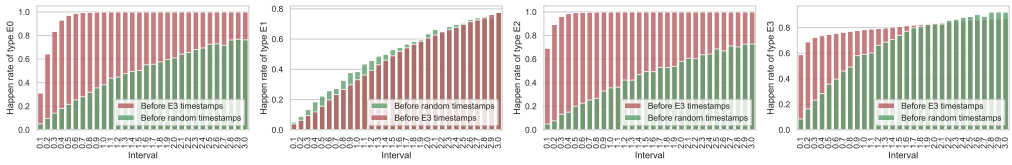
We utilize *tick.hawkes.SimuHawkesExpKernels* in the tick library to generate a sequence of events. We set time decay θ to 10, control the total simulation time as 10000, and then generate ~ 10000 events with timestamps for each synthetic dataset. The node features are created randomly, and the event feature is obtained by adding its ending node features.

Dataset statistics: As we focus on the prediction of E_3, we illustrate the statistics of E_0, E_1, E_2, and E_3 before E_3 happens. Specifically, we compare occurrence rates of all event types before an E_3 timestamp to those before a random timestamp. The x axis in Fig. 7 and Fig. 8 indicates selected intervals from 0.1 to 3.0. While the y axis indicates the happening rate of a specific event type in that interval before an E_3 or a random timestamp.

Take the Fig. 7(a) as an example, assuming the interval is 0.3, an E_0 event will happen with probability $\sim 82\%$ in the previous 0.3 time interval before an E_3 event, while the probability before a random timestamp is merely $\sim 16\%$. The other sub-figures in Fig. 7 and Fig. 8 illustrate happening rates of other event types.

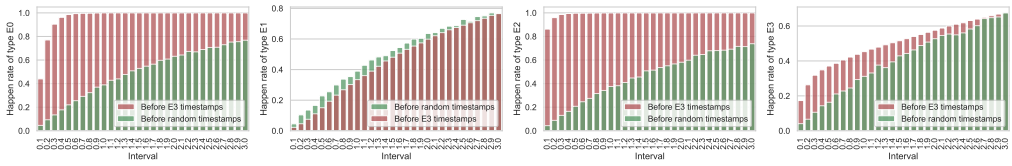
We can find that the happening rates of E_0 and E_2 are high before E_3 timestamps, especially when the time interval is small. It is the same as our setting, where E_0 and E_2 act as positive stimulus to E_3. Moreover, the happening rate of E_1 before E_3 is even smaller than that before random timestamps, indicating that E_1 will suppress the happening of E_3. Comparing Fig. 7(d) with Fig. 8(d), we could also conclude that in the Synthetic v2, E_3 itself will suppress E_3 because the happening rate of E_3 before E_3 in Fig. 8(d) is clearly smaller than that in Fig. 7(d).

In both Fig. 7 and Fig. 8, the distributions of previous events' happening rates are different for E_3 timestamps and random ones. Hence these signals are captured and utilized by target models to predict whether an E_3 will happen given previous events.



(a) E_0 (b) E_1 (c) E_2 (d) E_3

Figure 7: Statistics of the Synthetic v1.



(a) E_0 (b) E_1 (c) E_2 (d) E_3

Figure 8: Statistics of the Synthetic v2.

A.3 Details of Target Models

We list all the hyper-parameters of target models in Table 3. Note that *N degree* indicates the number of neighbour temporal events used for information aggregation in the message passing.

Besides, we also present average precision (AP) of well-trained target models on all datasets in Table 4. The results are averaged over three runs.

Table 3: Hyper-parameters of both models for all datasets

	TGAT		TGN	
	Real-world	Synthetic	Real-world	Synthetic
Hidden dimension	172	4	172	4
Aggregation layers	2	2	2	2
Attention heads	2	2	2	2
N degree	10	10	10	10
Memory dimension	-	-	172	4
Time dimension	172	4	172	4
Node feature dimension	172(zeros)	4	172(zeros)	4
Edge feature dimension	172	4	172	4
Training epoch	10	100	10	100
Learning rate	1e-4	1e-4	1e-4	1e-4

Table 4: Models' average precision (AP) for the inductive event prediction on all datasets.

	Wikipedia	Reddit	Simulate v1	Simulate v2
TGAT	0.9791(0.0000)	0.9750(0.0000)	0.9632(0.0005)	0.9641(0.0000)
TGN	0.9851(0.0000)	0.9664(0.0000)	0.9535(0.0000)	0.9687(0.0000)

A.4 Efficiency comparison with other baselines.

In this section, we compare the running time of all the methods for searching for a solution satisfying a fidelity threshold. The fidelity threshold is identical to that in Sec. 5.4.

The results w.r.t TGAT and TGN are listed in Table 5 and Table 6 respectively. Both tables indicate that T-GNNExplainer with navigator is much faster than T-GNNExplainer without navigator. The running time of T-GNNExplainer with navigator is acceptable in most cases. All the non-search based baselines achieve high efficiency because they conduct a one-pass inference or simply average internal weights. Even though T-GNNExplainer is slower than non-search based baselines, the AUFSC is 86% higher than the leading baseline averaged on all the datasets and models. Moreover, some baselines require retraining on new datasets, e.g., PG, which could be time-consuming once the datasets are large. Hence, considering the explanation quality and intrinsic characteristic of searching, the efficiency of T-GNNExplainer is reasonable and acceptable.

Table 5: Running time comparison of different methods on all the datasets for explaining an instance with TGAT. † indicates withholding the navigator.

	Methods/Time (s)	Wikipedia	Reddit	Synthetic v1	Synthetic v2
non-search based	ATTN	0.05	0.17	0.05	0.05
	PBONE	0.31	0.39	0.23	0.25
	PG	0.03	0.22	0.03	0.03
search based	T-GNNExplainer†	68.14	158.2	89.74	178.2
	T-GNNExplainer	20.38	28.2	14.49	12.5

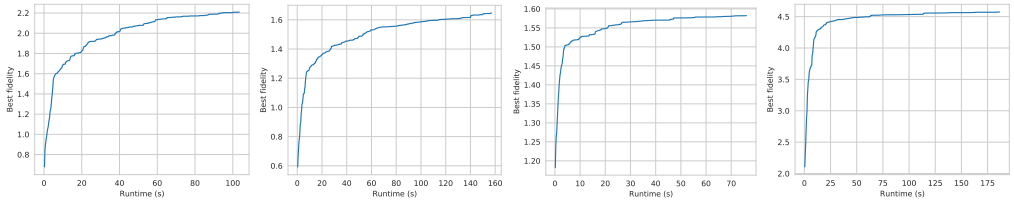
Table 6: Running time comparison of different methods on all the datasets for explaining an instance with TGN. † indicates withholding the navigator.

	Methods/Time (s)	Wikipedia	Reddit	Synthetic v1	Synthetic v2
non-search based	ATTN	0.04	0.16	0.03	0.03
	PBONE	0.17	0.31	0.14	0.17
	PG	0.03	0.14	0.10	0.09
search based	T-GNNExplainer†	20.50	40.26	31.95	56.10
	T-GNNExplainer	14.74	8.66	18.00	22.14

Moreover, we can deduce the complexity of our method. Since MCTS is an anytime algorithm, i.e., it can stop at any time based on the rollout limitation, the complexity is $\mathcal{O}(NDC)$. N is the

number of rollouts, D is the expansion depth of each rollout determined by the sparsity threshold, and C is a constant including inference time of the navigator, storing time of tree nodes, and other constant-time operations.

We plot runtime-fidelity curves of T-GNNExplainer for TGAT in Fig. 9 to better illustrate the trade-off between efficiency and solutions' quality. Fig. 9 reveals that the best fidelity of found solutions increases steeply at the beginning of searching and the marginal gain decreases with the increase of rollouts in all cases. It means that the method could find a reasonable solution without many rollouts. Hence, in practice, we could use a relatively small number (e.g., [100, 200]) to balance the efficiency and expected solution quality.

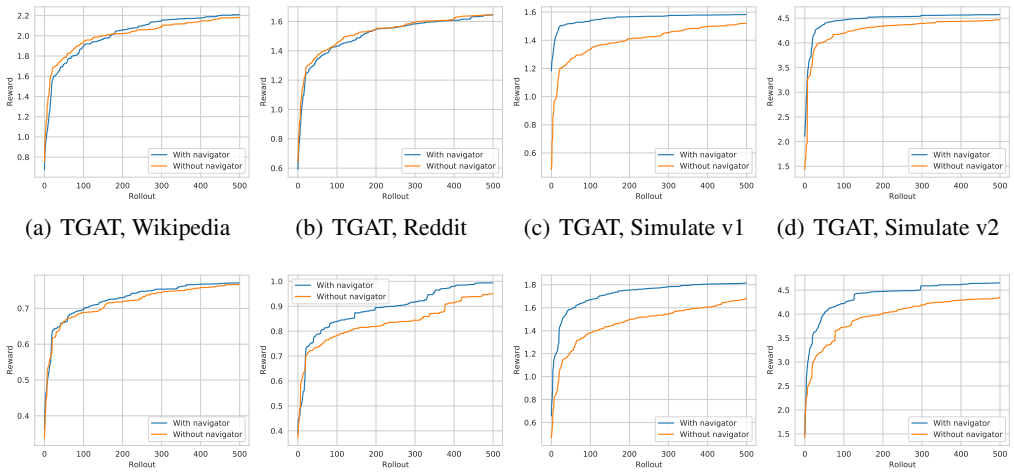


(a) TGAT, Wikipedia (b) TGAT, Reddit (c) TGAT, Simulate v1 (d) TGAT, Simulate v2

Figure 9: Runtime-fidelity tradeoff of TGAT on all the datasets.

A.5 Rollout-Reward comparison with or without the navigator

In this section, we investigate the effect of the navigator using reward-rollout curves for both target models on all datasets. The reward-rollout curve reflects the best solution's quality difference within a specific rollout limitation. Results in Fig. 10 show that *with navigator* outperforms *without navigator* in most cases, except for the TGAT&Reddit scenario, in which the target model and the navigator may not be trained well because of model capacity and the noisy characteristic of the Reddit dataset. The performance gap between *with navigator* and *without navigator* becomes more significant on synthetic datasets. Since target models achieve better prediction performance, the navigator can be trained more satisfactorily to capture events' importance as well.



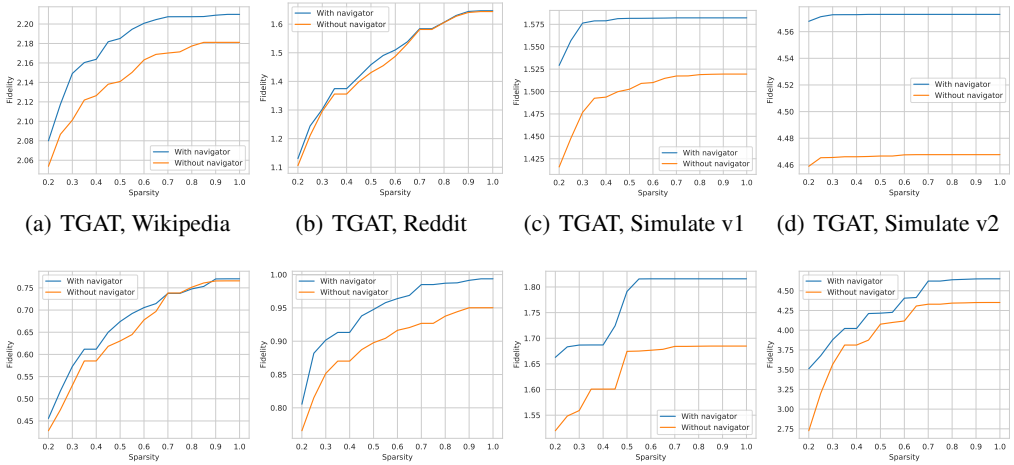
(a) TGAT, Wikipedia (b) TGAT, Reddit (c) TGAT, Simulate v1 (d) TGAT, Simulate v2

(e) TGN, Wikipedia (f) TGN, Reddit (g) TGN, Simulate v1 (h) TGN, Simulate v2

Figure 10: Reward-rollout comparison w/wo the navigator.

A.6 FIDELITY-SPARSITY COMPARISON WITH OR WITHOUT THE NAVIGATOR

In this section, we plot fidelity-sparsity curves to compare *with navigator* and *without navigator* from a different perspective. The fidelity-sparsity comparison reveals the best solution's quality difference under a specific sparsity threshold after the search terminates. In Fig. 11, we find that the fidelity gaps are generally consistent with those in Fig. 10, i.e., *with navigator* could achieve a higher fidelity than *without navigator* under a given sparsity threshold in most cases. We conclude Fig. 11 and Fig. 10 that the navigator could not only accelerate the search process, but also boost the quality of solutions under the limitation of rollouts and sparsity.



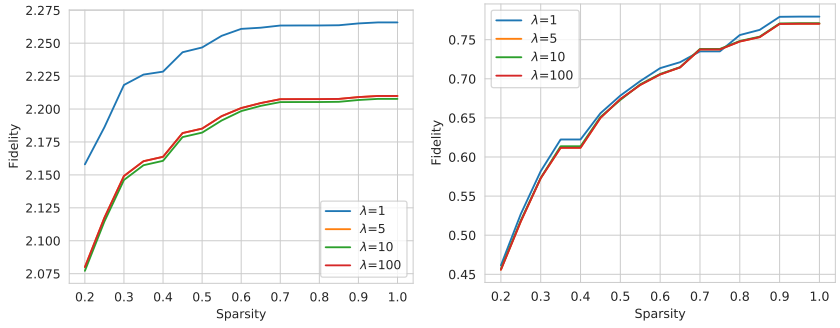
(a) TGAT, Wikipedia (b) TGAT, Reddit (c) TGAT, Simulate v1 (d) TGAT, Simulate v2

(e) TGN, Wikipedia (f) TGN, Reddit (g) TGN, Simulate v1 (h) TGN, Simulate v2

Figure 11: Fidelity-sparsity comparison w/wo the navigator.

A.7 HYPERPARAMETER ANALYSIS

In this section, we investigate the effect of the hyperparameter λ in Eq. 4. λ balances the exploitation and exploration in the search process, hence it influences searched solutions' quality as well. We conduct the experiment with both target models on the Wikipedia dataset, and λ is set to 1, 5, 10, and 100, respectively. The results are shown in Fig. 12. We find that a smaller λ is slightly better than a larger one in both scenarios, indicating that more exploitation is preferred in T-GNNExplainer because of the existence of the navigator. However, the absolute difference is insignificant compared with the fidelity scale. In practice, we can set the λ in the range $[1, 10]$ for better performance.



(a) TGAT, Wikipedia (b) TGN, Wikipedia

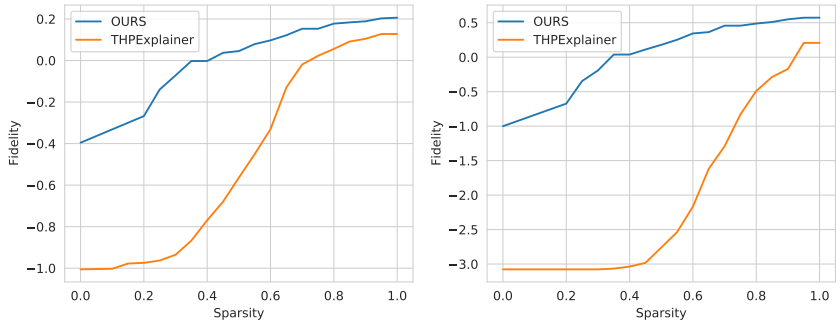
Figure 12: Hyperparameter analysis of λ for TGAT and TGN on Wikipedia.

A.8 COMPARISON WITH A SELF-INTERPRETABLE BASELINE

In this section, we compare T-GNNExplainer with a self-interpretable model named Transformer Hawkes Process (THP) Zuo et al. (2020), to investigate the proposed post-hoc explainer's explanation quality with a non-post-hoc one. The THP is based on the point process framework which utilizes a conditional intensity function to model previous events' influence on future ones. The THP adopts a transformer to learn a neural conditional intensity function $\Lambda_\theta(i, j, \delta t)$, where θ is the parameter set, i and j indicate event types, and δt represents a time interval. $\Lambda_\theta(i, j, \delta t)$ models the stimulus intensity of a past event with type i with time interval δt to the happening of event type j at the current timestamp. Since THP considers all event types' correlations, it cannot scale to real-world temporal graphs with tens of thousands of edges (i.e., types). We train and compare with THP on two synthetic datasets. The THP is trained to predict the next event's type and happening time. After training, we can compute each past event's $\Lambda_\theta(\cdot, \cdot, \cdot)$ as a score for interpretation. We denote the intrinsic interpretation method of THP as THPExplainer. Moreover, we regard the THP as a target model to be explained and use the prediction logits for the target event type E_3 to compute reward and run T-GNNExplainer. We compare the fidelity-sparsity curves of T-GNNExplainer and THPExplainer in Fig. 13, and compute Best Fid and AUFSC in Table 7. We find that T-GNNExplainer outperforms THPExplainer on all sparsity thresholds and the improvement is more significant for a smaller sparsity. From both Table 1 and Fig. 13, we could conclude that a search-based method could generally find solutions superior to conditional intensity scores or attentions. These scores are also sensitive to the model's performance. The performance may influence explanation quality as well. For example, THP has about 70% accuracy on simulated datasets because it models both time intervals and event types, which is challenging. TGAT/TGN achieves about 90% accuracy since they only model a binary task. More importantly, these scores may not completely and accurately reflect the decision logic of a complicated neural model. Clarifying the relationship between attention mechanism or conditional intensity scores and model-level interpretation may deserve further investigation.

Table 7: Best fidelity (↑) and AUFSC (↑) achieved by T-GNNExplainer and THPExplainer on synthetic datasets.

	Synthetic v1		Synthetic v2	
	Best Fid	AUFSC	Best Fid	AUFSC
THPExplainer	0.127	-0.485	0.207	-2.046
T-GNNExplainer	0.206	-0.006	0.573	0.021



(a) Simulate v1 (b) Simulate v2

Figure 13: Fidelity-sparsity comparison between T-GNNExplainer and THPExplainer on two synthetic datasets.