NETXPLAIN: REAL-TIME EXPLAINABILITY OF GRAPH NEURAL NETWORKS APPLIED TO NETWORKING

David Pujol-Perich[1], José Suárez-Varela[1], Shihan Xiao[2], Bo Wu[2], Albert Cabellos-Aparicio[1], Pere Barlet-Ros[1]
[1]Barcelona Neural Networking center, Universitat Politècnica de Catalunya., [2]Network Technology Lab., Huawei Technologies Co.,Ltd.

NOTE: Corresponding author: David Pujol-Perich, david.pujol.perich@upc.edu

Abstract – *Recent advancements in Deep Learning (DL) have revolutionized the way we can efficiently tackle complex optimization problems. However, existing DL-based solutions are often considered as black boxes with high inner complexity. As a result, there is still certain skepticism among the networking industry about their practical viability to operate data networks. In this context, explainability techniques have recently emerged to unveil why DL models make each decision. This paper focuses on the explainability of Graph Neural Networks (GNNs) applied to networking. GNNs are a novel DL family with unique properties to generalize over graphs. As a result, they have shown unprecedented performance to solve complex network optimization problems. This paper presents NetXplain, a novel real-time explainability solution that uses a GNN to interpret the output produced by another GNN. In the evaluation, we apply the proposed explainability method to RouteNet, a GNN model that predicts end-to-end QoS metrics in networks. We show that NetXplain operates more than 3 orders of magnitude faster than state-of-the-art explainability solutions when applied to networks up to 24 nodes, which makes it compatible with real-time applications; while demonstrating strong capabilities to generalize to network scenarios not seen during training.*

Keywords – AI/ML for networks, explainability, graph neural networks

1. INTRODUCTION

In recent years, Deep Learning (DL) has revolutionized the way we are able to solve a vast number of problems by finding meaningful patterns on large amounts of data. This acquired knowledge then enables us to make highly accurate predictions, leading to systematically outperforming state-of-the-art solutions in many different problems [1, 2]. However, in the field of networking, DL-based techniques still pose an important technological barrier to achieve market adoption. In general, Machine Learning (ML) solutions provide probabilistic performance guarantees, which typically degrade as the data deviates from the distribution observed during training. Moreover, neural networks have very complex internal architectures, often with thousands or even millions of parameters not interpretable by humans. As a result, they are treated as black boxes [3]. This limits the viability of these solutions to be applied to networks, as these are critical infrastructures where it is essential to deploy fully reliable solutions. Otherwise, a potential misconfiguration could lead to temporal service disruptions with serious economic damages for network operators.

In this vein, we do need mechanisms that can delimit the safe operational ranges of DL models. This makes it fundamental to *understand why and in what situations a DL-based solution can fail*. This can be achieved by producing human-readable interpretations of the decisions made by these models (e.g., interpret a routing decision given a traffic matrix and a network topology). This would not only enable us to achieve more mature and reliable DL solutions but also to enhance their performance by making ad-hoc adjustments for a particular network scenario (e.g., hyper-parameter tuning).

In this context, explainability solutions [4] have recently emerged as practical tools to interpret systematically the decisions produced by DL models. Particularly, these recently proposed solutions analyze trained DL models from a black-box perspective (i.e., they only analyze their inputs and outputs) and aim to discover which elements mainly drive the output produced by these models. As a result, they can eventually determine what are the most critical input elements to reach the final decisions. These kinds of techniques have been intensely examined in the field of computer vision, showing promising results [5].

At the same time, the last few years have seen the explosion of Graph Neural Networks (GNNs) [6], a new neural network family that has attracted large interest given its numerous applications to different fields where the information is fundamentally represented as graphs (e.g., chemistry [7], physics [8], biology [9], information science [10, 11]). This newly introduced mechanism has proven, to date, to be the only DL technique capable of generalizing with high accuracy to graphs of different sizes and structures not seen during the training phase.

In this context, GNNs have shown good properties to be applied in the field of computer networks, as many key components in network control and management problems are fundamentally represented as graphs (e.g., topology, routing). Indeed, we have already witnessed some successful GNN-based applications to network modeling and optimization [12, 13, 14, 15]. However, the fact that we are not able to understand the inner architecture of GNNs presents nowadays a major barrier that may hinder its adoption in real-world networks.

Explainability of GNNs has been recently explored in two main works. A first work emerging from the ML community [10] analyzes a well-known GNN model applied to a chemistry problem to quantify the impact of the different input elements (atoms, bonds) on the final model predictions (molecular properties). Likewise, the networking community has made a first attempt to apply a similar solution to several network optimization use cases [3]. However, both solutions are based on costly iterative optimization algorithms that need to be executed for each sample on which we want to obtain interpretations. Hence, they do not meet the requirements to make comprehensive analysis over large data sets and, more importantly, to be used in real-time applications. To address these limitations, this paper proposes NetXplain, a novel real-time *explainability solution for GNNs*. NetXplain introduces a novel approach where we use a GNN that learns, from Tabula Rasa, how to interpret the outputs of another GNN trained for a specific task (e.g., routing optimization). NetXplain produces human-understandable interpretations of GNNs comparable to those of state-of-the-art solutions [10, 3]. However, it makes this at a much more limited cost. In our evaluation, we apply NetXplain to RouteNet [12], a GNN model that predicts the per-path delay given a network snapshot as input (i.e., topology + routing + traffic matrix). To this end, we first train NetXplain on a data set with samples produced by Metis [3]. This training is done over a data set of limited size –5 to 10% of the original data set used in RouteNet [12]. Then, we validate the generalization power of our GNN-based method, by applying it to network scenarios fundamentally different from those seen during training. The evaluation results reveal the feasibility to train NetXplain over a small dataset produced by costly explainability solutions (e.g., [10, 3]), and be able to apply it over a wide variety of network scenarios. This eventually enables us to meet the needed requirements to make a comprehensive analysis of the safe operational range of GNN solutions at a limited cost. In this context, we show that NetXplain far outperforms state-of-the-art algorithms in terms of computational cost, running more than 3 orders of magnitude faster on average than Metis [3] when applied to samples of three real-world network topologies up to 24 nodes.

As an example, this explainability solution can be used as follows: given a GNN-based solution and a network scenario, NetXplain points to the particular network elements (e.g., devices, links, paths) that mostly affected the output decisions of the GNN model. This can be helpful for many different use cases, including: (i) test & troubleshooting of GNN-based solutions, (ii) reverse engineering, or (iii) improving network optimization solutions.

The remainder of this paper is structured as follows. First, Section 2 introduces the fundamental principles of GNNs and their application to networking. Then, Section 3 presents the related work on explainability for GNNs. In Section 5 we describe NetXplain, the proposed explainability solution. Afterward, Section 6 presents an evaluation of the accuracy and cost of NetXplain with respect to the state of the art. Finally, Section 7 presents a discussion on possible applications of the proposed explainability method, and Section 8 concludes the paper.

2. BACKGROUND

2.1 Graph neural networks

Graph neural networks are a novel neural network family designed to operate over graph-structured data, by capturing and modeling the inherent patterns in a graph. This has resulted in an unprecedented predictive power in many applications where data is structured as graphs. Despite the several variants of GNNs introduced in recent years, in this paper we focus on *Message-Passing Neural Networks (MPNN)* [7], as they represent a generic GNN framework.

MPNN operates over a graph *G*, in which nodes $v \in G$ are characterized with some initial features X_v. First, the hidden-state h_v^0 of each node $v \in G$ are initialized using their input node features X_v. Once each element v of the graph has its hidden-state h_v^0 initialized, they proceed to the message-passing phase, which shall be repeated a given number of times T. Fig. 1 illustrates how the message passing phase works. In each iteration t of the algorithm, every node v receives a message from each of its neighbors $u \in N(v)$. In MPNN, messages are generated using a function $m(\cdot)$ computed with the hidden state of the neighbor node. Then, once every node v has received the messages from its immediate neighbors, these messages are combined with an aggregation function $a(\cdot)$ producing a fixed-size output (e.g., an element-wise summation).

Finally, the algorithm reaches the update phase, in which nodes use the aggregated information received from their neighbors to update their own hidden states via the update function $u(\cdot)$.

Formally, the message passing at a given iteration t is defined as:

$$m_{v,j} = m(h_v^t,\ h_j^t,\ e_{v,j}) \quad (1)$$

$$M_v^{t+1} = a(\{m_{v,j} \mid j \in N(v)\}) \quad (2)$$

$$h_v^{t+1} = u(h_v^t,\ M_v^{t+1}) \quad (3)$$

In these equations, functions $m(\cdot)$ and $u(\cdot)$ can be computed through a universal function approximator, such as neural networks (e.g., feed-forward NN or recurrent NN). After T message passings, the hidden states of nodes typically converge to some fixed values [6]. Thus, these final hidden states pass through a readout function $r(\cdot)$ that computes the output of the GNN model. $r(\cdot)$ automatically learns the mapping from hidden-state representations to the output labels of the model y:

$$\hat{y} = r(h_v^T \mid v \in V) \quad (4)$$

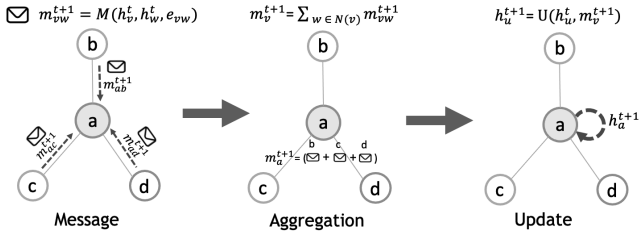


Fig. 1 – Message-passing phase: (left) Message, (mid) aggregation and (right) update.

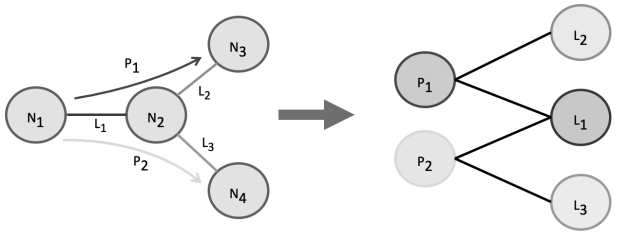


Fig. 2 – Transformation from the physical network scenario to the graph representation of RouteNet.

Such a function $r(\cdot)$ can also be implemented as a neural network, typically a feed-forward NN, and can be used to produce either node-level predictions by processing individually each node hidden state, or make global predictions of the graph by combining all the hidden states. In this latter case, hidden states are typically aggregated (e.g., element-wise sum) before they are introduced into the readout function.

This technology has proven to generalize successfully over graphs of different sizes and structures, which was not possible with traditional neural network architectures (e.g., feed-forward NN, convolutional NN, recurrent NN).

2.2 Graph neural networks applied to networking

The strong generalization capabilities of GNN over graphs make these models interesting for applications in the networking field since the most natural way to formalize many network control and management problems involves the use of graphs (e.g., topology, routing, interflow dependencies) [3]. Recently, several GNN-based solutions have been proposed to tackle different use cases in the field of computer networks (e.g., network modeling [12, 16], automatic routing protocols [13]). In this section, for illustrative purposes, we focus only on RouteNet [12], as it is quite representative of how GNN-based solutions represent and process network-related data to solve complex problems.

RouteNet targets the problem of modeling the per-path QoS metrics (e.g., delay, jitter) of a computer network. For this purpose, a network snapshot is provided as input: a network topology, a routing configuration, and a traffic matrix. To this end, this model makes a transformation of the physical network scenario into a more refined graph representation in which physical and logical elements are explicitly represented –*paths* and *links* in this case. More specifically, every *link* of the physical network topology is transformed into a node in the input graph of the GNN. Likewise, each source-destination path is also converted into a node. Finally, edges connect *links* with *paths* according to the routing configuration. Thus, each path is connected to those links that it traverses given the input routing scheme. This process is illustrated in Fig. 2, where we can observe how a physical network scenario with two paths and three links is transformed into the input graph of RouteNet. This graph representation enables us to model the complex relationships between the state of paths and links, and how they relate to the output per-path performance metrics (e.g., delay).

In this regard, applying explainability over this model would enable us to identify the most critical edges of its internal graph (i.e., path-link relations). We refer to critical edges as the set of path-link pairs that better explain the QoS metrics obtained by the model. Thus, with this solution, we can extract relevant knowledge of the processing made by the GNN given a network scenario, which can have many diverse applications, as later discussed in Section 7.

3. RELATED WORK

Recent years have attracted increasing interest in producing explainability solutions for neural network models (e.g., *Convolutional Neural Networks* [5]). Despite this, explainability techniques for GNN have been scarcely explored so far. In this context, GNNExplainer [17] is, to the best of our knowledge, the first proposal approaching this problem.

GNNExplainer is given as input a target GNN model and a sample graph $G = (V, E)$, with input features F. GNNExplainer, then, outputs a subset containing the connections $E' \subset E$ and the node features $F' \subset F$, that affect most critically the output of the target GNN (see Fig. 3). This is done by computing a set of weights W, formally defined in Eq. *(5)*, that represents how critical are the pair-wise connections of input graphs to the prediction accuracy of the target GNN.

$$W = \{w_{i,j} \mid (i,j) \in E\} \qquad (5)$$

Particularly, the most relevant connections are those that have more impact on the loss function used to train the model (e.g., mean squared error for regression tasks). The number of relevant connections produced by the algorithm can be tuned by setting a threshold on the resulting weights $w_{i,j} \in W$.

Overall, GNNExplainer is a generic solution proposed from the ML community that targets only at producing explainability representations of GNNs used for global graph classification, node-level classification, or link prediction. However, this solution does not support GNN-based models used for regression. In this context, a posterior solution proposed from the networking community presents Metis [3], a similar approach adapted to GNN models trained for regression problems, particularly showcasing its use in several networking applications.

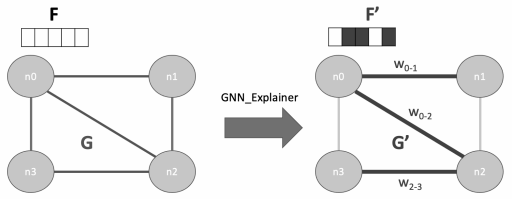


Fig. 3 – Schematic description of explainability solutions for GNN (e.g., GNNExplainer)

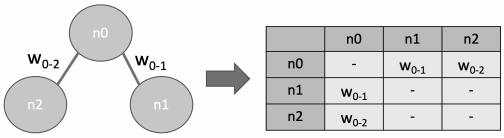


Fig. 4 – Explainability mask of an input graph.

Although GNNExplainer [17] and Metis [3] are able to produce quality explainability solutions for a vast range of problems, both have an important limiting factor. To compute E' and F' for each input sample, these solutions use an iterative convex optimization method, which is very time-consuming. For instance, producing a single explainability solution can take up to hundreds of seconds in scenarios with topologies between 14 and 24 nodes, as shown later in Section 6. This fact arguably prevents these methods to be applied for real-time operation. Moreover, their high cost makes them impractical to perform a comprehensive test analysis of GNN-based solutions, covering a wide range of network scenarios, before these tools are released to the market.

4. PRELIMINARIES

This section firstly introduces a detailed description of the representations produced by graph-based explainability methods, which are commonly referred to as *explainability masks*. Then, we present the general overview on how state-of-the-art explainability solutions produce *explainability masks* on graphs.

4.1 Explainability mask

We refer to the *explainability mask* as an $n \times n$ matrix that defines the relevance of each connection of an input graph $G = (V, E)$ on the output produced by the target GNN, where $n = |V|$. This mask enables us to interpret which are the main graph elements that affect most the predicting power of the GNN in each case.

Formally, given an input graph $G = (V, E)$, state-of-the-art explainability methods aim to produce an explainability mask $W \in \{0, 1\}^{n \times n}$, where each element defines a weight $W_{i,j}$ indicating the importance of the connection between node i and node j on the overall accuracy of the target GNN. Fig. 4 illustrates how the explainability mask is built from an input (undirected) graph G. Particularly, this matrix contains a weight for each pair (i, j) connected in the graph.

Note that when applying GNN to network-related problems, input graphs G may contain a wide variety of elements and connections that do not necessarily correspond to physical network elements (e.g., forwarding devices, links). For instance, some proposals like [12, 16] introduce complex hypergraphs including logic network entities (e.g., end-to-end paths).

4.2 Generating explainability masks

Current explainability solutions are based on iterative optimization methods, which work as follows:

Given a target GNN and an input graph $G = (V, E)$, explainability algorithms apply an iterative (costly) gradient descent method to compute the explainability mask W that best explains the accuracy of the model (i.e., it defines the set of weights W (see Eq. *(5)*)) that represents the impact of each graph edge on the loss function of the target GNN. More specifically, the calculation of the explainability mask is driven by the loss function of Eq. *(6)*, which depends on three factors: (i) predictive loss, (ii) entropy of the values in the mask, and (iii) L1 regularization computed over the mask. The predictive loss quantifies how the accuracy of the target GNN (Y_I) degrades when weighting the hidden states according to W (Y_W). Note that the predictive loss function greatly depends on the specific problem we aim to solve (e.g., regression or classification). The entropy factor (Eq. *(7)*) controls the trade-off between too homogeneous or too sparse values in the resulting mask W. Finally, the $L1$ regularization controls the number of connections that will have high values. More in detail, as the regularization factor has more importance, the mask will be driven towards having less critical connections (i.e., less high-value weights), which can be more useful for human interpretability. Moreover, notice that both entropy and regularization loss are weighted according to two hyperparameters (i.e., α, β) that can be fine-tuned according to the problem's needs.

Through a gradient descent method, these algorithms gradually converge to the optimal mask W^* that minimizes the loss function $\ell(W)$.

$$\ell(W) = P(Y_I, V_W) + \alpha H(W) + \beta ||W||_{L1} \quad (6)$$

$$H(W) = -\sum_{i,j} (W_{i,j} \log(W_{i,j}) + (1 - W_{i,j}) \log(1 - W_{i,j})) \quad (7)$$

5. NETXPLAIN: PROPOSED EXPLAINABILITY METHOD

In this section, we introduce NetXplain, a novel explainability method for GNN, compatible with real-time operations, that addresses the performance limitations of existing solutions (Section 3). NetXplain is able to produce the same output as state-of-the-art solutions, based on costly iterative optimization algorithms [17, 3], while operating at a much limited cost (at the scale of a few milliseconds in our experiments in Section 6). This not only enables us

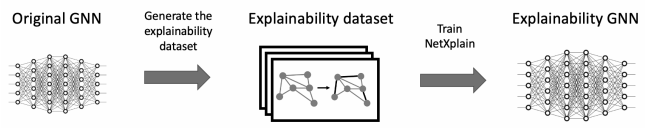


Fig. 5 – High-level workflow of NetXplain.

to perform real-time troubleshooting of GNN-based solutions applied to networks but also opens the possibility of combining these solutions with automatic optimization algorithms (e.g., local search, reinforcement learning) to solve more efficiently online optimization problems, as discussed later in Section 7. To this end, *NetXplain uses a GNN that learns how to interpret a target GNN model* that has been trained for a particular task. As shown in Fig. 5, the proposed GNN-based solution is trained with an *explainability data set* generated by an iterative optimization algorithm [3] and, once trained, the resulting model can make one-step explainability predictions for each input sample of the target GNN. Note that thanks to the generalization capabilities of GNN over graph-structured information, once NetXplain is trained over a particular target GNN solution, it can be applied to different input graphs not included in the training data set. In practice, when applied to GNN-based networking solutions, NetXplain is able to generalize to network scenarios with topologies of variable size and structure not seen in advance, as shown later in the experiments of Section 6. The following subsections describe in more detail the main components of this solution.

5.1 Explainability data set

To train NetXplain, we first need to generate the new explainability data set, which we refer to as A. To this end, we first randomly sample a subset $D' \subseteq D$, where D is the original data set used to train the target GNN. Given this subset D', we now target the problem of producing, for each input graph $G \in D'$, its associated explainability mask W_G when applied to the target GNN. Note that this process is made from a black-box perspective (i.e., the explainability mask interprets the relevance of the input graph connections by analyzing the input-output correlations in the target GNN). For this task we can use specific state-of-the-art iterative optimization algorithms, introduced in Section 3, and further described in Section 4.2, depending on the particularities and the purpose of the target GNN (e.g., regression, classification).

Thus, we apply the process described in Section 4.2 for each of the samples $G \in D'$. Hereby, we eventually obtain the final explainability data set A, formally defined in Eq. *(8)*, which maps each of the selected graphs to its corresponding optimal mask W^*_G.

$$A = \{(G, W^*_G) \mid G \in D'\} \qquad (8)$$

Note that due to the high cost of computing the explainability data set, it is crucial to ensure that $|D'| << |D|$. For instance, in our experiments, we observe that NetXplain is able to converge to a valid solution using only 5-10% of the samples of the original data sets. Consequently, the cost of generating the explainability data set becomes much more affordable than applying the iterative optimization algorithm over all the samples of D.

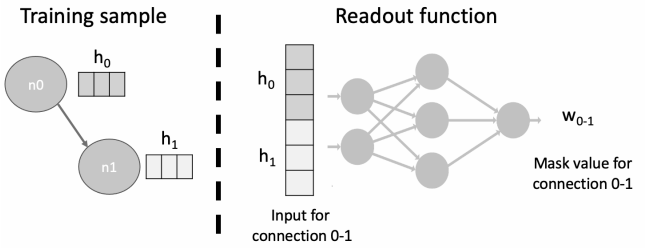


Fig. 6 – Adaptation of the readout function in NetXplain to produce the explainability mask.

5.2 Training the explainability GNN

Finally, we propose the use of an independent GNN (NetXplain) to learn how to predict explainability masks W_G over the target GNN for an input graph $G = (V, E)$.

First, we must define the underlying architecture of the NetXplain GNN, which we use for training. Particularly, we mostly keep the same architecture of the target GNN. The intuition behind this decision is that the complexity for the target GNN to learn how to make its output predictions should be similar to solving the explainability problem over that GNN (i.e., explaining which connections affected most such predictions). However, it is needed to make a minor change on the readout function $r(\cdot)$, in order to adapt it to produce the explainability mask M_G. As illustrated in Fig. 6, for every edge $(i, j) \in E$, we concatenate their final hidden-state vectors after the message passing phase is finished (i.e., $h_i^T \mid\mid h_j^T$) and this is passed as input to $r(\cdot)$, which predicts the mask weight for that edge $W_{i,j}$. Note that this operation can be computed in parallel for each node pair $(i, j) \in E$ of the input graph.

A key aspect of our proposal is to reduce as much as possible the subset of samples randomly selected (D') used to generate the samples of the explainability data set (A), which are finally used to train NetXplain's GNN. The reason is that typically producing explainability masks for all the samples of the original data set D may be too costly with state-of-the-art explainability solutions. To achieve this, we follow a transfer learning approach. Particularly, we first initialize the explainability GNN with the same internal parameters (i.e., weights and biases) of the target GNN model, except for the readout function, whose implementation differs as explained before. This enables us to effectively initialize the explainability GNN model, as the message-passing functions of this GNN are expected to be close to those of the target GNN (e.g., similar graphs and feature distributions). Thus, during training, the main adjustment should be made over the readout function. To this end, we finally train the explainability model with a reduced explainability data set A generated by a reference explainability algorithm, and this enables us to learn how to produce accurately explainability masks.

5.3 Generalization power of NetXplain

By analyzing the training process of NetXplain, we identify the generation of the data set A as the most computationally costly task, even when considering that the number of selected samples of D is only a small portion of the original data set D.

Note that our proposal aims to learn how to explain potentially any sample that our target GNN could face during operation. This motivates our choice of using a GNN to explain a target GNN. To the best of our knowledge, GNNs are the only technique that offers high generalization power over graph-structured data. As a result, once trained, the GNN explainability model generalizes to network scenarios not present in its training data set A. This means that NetXplain's GNN can be trained over a small data set to make predictions of the critical connections from the perspective of the target GNN and, once trained, it can predict in one step these critical connections over arbitrary network scenarios (e.g., topologies of variable size and structure). All this while offering an accuracy comparable to state-of-the-art costly solutions.

6. EVALUATION

In this section, we first evaluate the accuracy of the predictions made by NetXplain with respect to the state-of-the-art solutions (Metis [3]). Second, we quantify the speed-up when using NetXplain compared to Metis. In our experiments, we train an explainability model that makes interpretations over RouteNet [12], a GNN model used to make QoS inference in networks, previously introduced in more detail in Section 2.2.

All these experiments are evaluated over the same data sets used in RouteNet [12], which are publicly available at [18].

6.1 Generating the explainability model

First, we need to generate the explainability data set and define an architecture for the explainability GNN model:

6.1.1 Explainability data set

To train a NetXplain explainability model for RouteNet we first need to generate the explainability data set A (Section 5.1). In this case, we generate this data set using Metis [3].

To this end, we first train RouteNet as the target GNN model, using 300k samples simulated in the NSFNet network, including scenarios with various routing configurations and traffic matrices [18].

Before generating the explainability data set A, we randomly sample a subset $D' \subseteq D$ from the original data sets [18]. Note that the different experiments made in this section use different subsets D' to generate the explainability data sets A, finally used to train the NetXplain's GNN models. This is then specified in the respective sections. In general, our experimentation shows that this subset D' needs only $\approx 5\%$ of samples randomly extracted from the original data set D (i.e., approximately 15k samples) to ensure that NetXplain learns properly.

Afterward, we generate with Metis the final explainability data set A, as described in Section 5.2. In this process Metis maps each of the selected samples $G \in D'$ to its corresponding mask W_G, using as a target GNN the RouteNet model previously trained on samples of NSFNet. Note that Metis [3] is an iterative optimization algorithm. Hence, we limit it to run 2,000 iterations per sample, after observing this was sufficient to ensure convergence.

Finally, to train our NetXplain model, we make a random split of the explainability data set A (80%, 10%, and 10%) to produce the training, validation, and test data sets respectively.

Algorithm 1: Architecture of the NetXplain's explainability GNN applied to RouteNet

input : $\mathbf{x}_p$, $\mathbf{x}_l$,$\mathcal{R}$
output: h_v^T, h_e^T, h_p^T, W

begin
  // Initialize states of paths and links
  foreach $p \in \mathcal{R}$ **do** $h_p^0 \leftarrow [x_p, 0 \dots, 0]$;
  foreach $l \in \mathcal{N}$ **do** $h_l^0 \leftarrow [x_l, 0 \dots, 0]$;
  for $t = 1$ *to* T **do**
    // Message passing from links to paths
    foreach $p \in \mathcal{R}$ **do**
      $m_p^t = \{h_l^{t-1} \mid l \in p\}$
      $h_p^t \leftarrow RNN_p(h_p^{t-1},\ m_p^t)$
    end
    // Message passing from paths to links
    foreach $l \in \mathcal{N}$ **do**
      $m_l^t \leftarrow \sum_{p:l \in p} h_p^t$
      $h_l^t \leftarrow RNN_l\,(h_l^{t-1},\ m_l^t)$
    end
  end
  // Readout function
  foreach $p \in \mathcal{R}$ **do**
    foreach $l \in p$ **do**
      $q_{l,p} \leftarrow (h_l^T \mid h_p^T)$
      $W_{l,p} \leftarrow r(\ \mathbf{q}_{l,p}\ )$
    end
  end
end

6.1.2 Architecture of the explainability GNN

As previously mentioned in Section 5.2, we use for the explainability GNN a similar architecture to the target GNN, RouteNet [12] in this case. The only change introduced with respect to the original formulation of RouteNet is in the readout function. Algorithm 1 provides a detailed description of the NetXplain's explainability GNN when applied to RouteNet (see scheme of Fig. 2). In this case, the readout function outputs a weight $W_{l,p}$ for each link-path connection (l, p). To this end, we concatenate the corre-

sponding hidden states of the *link* (h_l) and the *path* (h_p), and introduce this as input of the readout function. Thus, the resulting weight $w_{l,p}$ can be interpreted as quantifying the importance for RouteNet of a particular src-dst path p as it passes through a certain link l of the network.

6.2 Evaluation of the accuracy

We evaluate the accuracy achieved by the NetXplain model on samples simulated in three real-world topologies [18]: *NSFNet* (14 nodes), *GEANT2* (24 nodes), and *GBN* (17 nodes). Concretely, for each topology we randomly pick 1,000 samples (with different routing configurations, and traffic matrices), and produce explainability masks with the NetXplain GNN model described in Section 6.1.2. Fig. 7 depicts the Cumulative Distri- bution Function (CDF) of the relative error produced by NetXplain's predictions with respect to those obtained by Metis [3], acting as the ground truth. We observe that our explainability model achieves a Mean Relative Error (MRE) of 2.4% when it is trained and evaluated over explainability data sets A with samples of the NSFNet topol- ogy (14 nodes). We then repeat the same experiment training and evaluating the model with samples of Geant2 (24 nodes), and obtain an MRE of 4.5%. Note that de- spite NetXplain's GNN being trained and evaluated over samples of the same topology, the network scenarios (i.e., routing and traffic matrices) are different across the train- ing and evaluation samples, which means that the input graphs seen by the GNN in the evaluation phase are dif- ferent from those observed during training. Finally, we further test the generalization capabilities of NetXplain by training the explainability GNN with samples from *NSFNet* and *GEANT2*, but in this case, we evaluate the model on samples of a different network: *GBN* (with 17 nodes). As a result, NetXplain achieves an MRE of 11%over this network topology unseen in advance (dashed line in Fig. 7). All these values are in line with the general- ization results already observed in the target GNN model (RouteNet [12]).

These results together show that using NetXplain we can achieve a similar output to a state-of-the-art solution based on iterative optimization (Metis [3]), even when our solution was tested over network scenarios not seen during training.

Table 1 – Execution time of NetXplain with respect to Metis, evaluated on three real-world network topologies

Topology	Method	Mean (s)	Std deviation (s)
NSFNet	Benchmark (Metis)	98.139	2.455
	NetXplain	0.012	0.001
GBN	Benchmark (Metis)	150.83	1.79
	NetXplain	0.0214	0.005
GEANT2	Benchmark (Metis)	191.46	2.76
	NetXplain	0.029	0.002

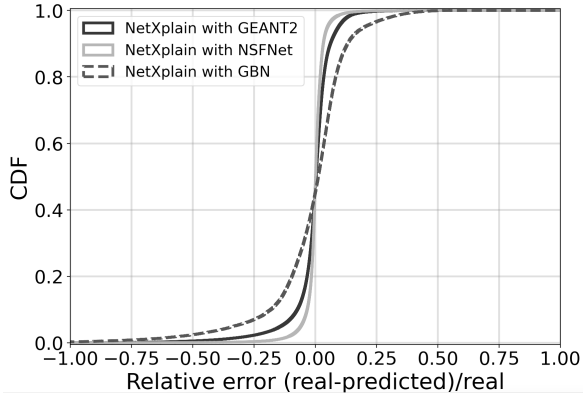


Fig. 7 – CDF of the relative error of NetXplain evaluated on three real-world network topologies.

6.3 Evaluation of the execution cost

In this section, we evaluate the computational time of NetXplain with respect to the original solution used to generate the explainability data set (Metis [3]). We thus measured the time to produce the output explainability masks using both solutions. This was done by randomly selecting 500 samples from each of the three topologies previously used in the experiments of Section 6.2: *NSFNet* (14 nodes), *GEANT2* (24 nodes), and *GBN* (17 nodes) [18]. Table 1 shows the execution times per sample during inference (in seconds), differentiated over the three considered data sets. Note that both solutions were executed in CPU and in equal conditions (they were applied over the same samples). We can observe that Metis takes ≈98 seconds on average to produce an explainability mask for an input sample of *NSFNet* (14 nodes). In contrast, NetXplain produced each mask in 12 ms on average. This constitutes a mean speed-up of ≈8,178x in the execution time. As we can observe, similar results are obtained for the samples of the other two network topologies, resulting in an average speed-up of ≈7,200x across all the topologies (i.e., more than 3 orders of magnitude faster).
This shows the benefits of NetXplain with respect to state-of-the-art solutions, as it can be used to make extensive explainability analysis at a limited cost (e.g., to delimit the safe operational range of the target GNN). More importantly, its operation at the scale of milliseconds makes it compatible with real-time networking applications.

7. DISCUSSION ON POSSIBLE APPLICATIONS

As previously mentioned, GNNs have been mainly leveraged for global network control and management tasks [3], as these scenarios typically involve modeling complex (and mutually recursive) relationships between different network elements (e.g., devices, links, paths) to then produce the system's output (e.g., end-to-end QoS metrics [12], routing decisions [15, 13]). In this section, we draw a taxonomy with three main use case categories where the application of GNN-based explainability solutions can be especially beneficial (Fig. 8):

(i) test & troubleshooting, (ii) reverse engineering, and (iii) improving optimization tasks. Particularly, we put the focus on the advantages of leveraging the fast and low-cost interpretations of NetXplain with respect to state-of-the-art explainability methods.

7.1 Test & troubleshooting

In order to achieve GNN-based products for networking, we need guarantees that they will work optimally when deployed in real-world networks. In this context, vendors would typically need to make extensive tests to their GNN solutions to check how they respond under different network conditions. Using NetXplain would enable us to collect human-readable interpretations of the internal data processing made by GNNs. For instance, if we have a GNN model that performs traffic engineering, we can identify the network elements that mainly drive the decisions made by the model, which are given by the explainability mask of NetXplain, and then observe if the properties of the selected elements are consistent across similar network scenarios. This would be a good indicator that the model generalizes well and, consequently, it is reliable for deployment. In this vein, with extensive testing we can find the safe operational range of models, which is essential for vendors to offer guarantees before selling their products (e.g., this product works optimally in networks up to 100 nodes and link capacities up to 40Gbps). Otherwise, operators would not take the risk of deploying such solutions on their networks, as they are critical infrastructures where misconfigurations are not acceptable. In this context, making such a comprehensive analysis using state-of-the-art solutions would result in large costs for vendors; while the limited cost of NetXplain would enable us to reduce dramatically both the cost and the time needed before releasing the product to the market.

Moreover, this testing process would enable us to troubleshoot GNN models by identifying particular scenarios where they are not focusing on the expected elements, or simply their behavior is not consistent with other similar scenarios. In this context, understanding *where and why* a model failed is crucial to refine it through an iterative training-testing process. For instance, it can help find deficiencies in the internal message-passing architecture that make the model less robust to particular network scenarios or identify a lack of samples in the training data sets.

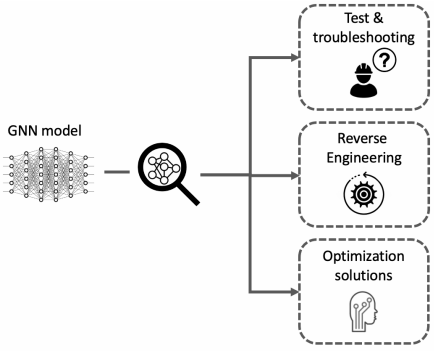


Fig. 8 – Possible applications of NetXplain.

7.2 Reverse engineering

One interesting application of ML-based solutions is to extract information about the knowledge learned during the training phase (i.e., reverse engineering). In this context, the explainability interpretations produced by NetXplain would enable us to understand what are the main network elements that GNNs consider before making their decisions. As a result, this may enable us to obtain non-trivial knowledge that can be leveraged to then design and implement efficient optimization algorithms and/or heuristics with deterministic and predictable behavior. These kinds of solutions are often perceived as more valuable by network operators, as nowadays there is a certain skepticism on applying ML-based solutions to real-world networks, mainly due to the critical nature of these infrastructures and the probabilistic guarantees typically offered by ML solutions.

7.3 Improving network optimization solutions

Network optimization problems often require dealing with very large spaces of possible actions (e.g., all the valid src-dst routing combinations in a network). As a result, optimization tools can only evaluate a small portion of configurations before they make a final decision. Thus, the exploration strategy used by these tools has a critical impact on the performance they can eventually achieve. In this context, explainability methods can provide meaningful interpretations of the current network state that can be useful to guide more efficiently optimization algorithms (e.g., reinforcement learning [15], local search [19]). For instance, using a NetXplain model trained over RouteNet, as the one of Section 6, would enable us to point to critical paths and links that are mostly affecting the network performance (e.g., end-to-end delays). This could be highly beneficial for optimization algorithms to explore alternative configurations targeting specifically these critical points (e.g., re-routing specific paths to avoid the critical points selected by NetXplain). In this context, computational efficiency is a must for optimization tools, as it directly affects the number of configurations that can be evaluated before producing the final decision. Thus, counting on solutions compatible with real-time operation, like NetXplain, offers an important competitive advantage with respect to state-of-the-art explainability solutions.

8. CONCLUSIONS

In this paper, we proposed NetXplain, an efficient explainability solution for Graph Neural Networks (GNNs). Particularly, this solution uses a GNN that *learns* how to produce accurate interpretations over the outputs produced

by another GNN model. In contrast to state-of-the-art solutions based on costly optimization algorithms, the proposed solution can be integrated into network control and troubleshooting systems operating in real time. We tested NetXplain over RouteNet, a GNN model that predicts per-source-destination delays in computer networks, and showed that our solution can produce an output equivalent to state-of-the-art solutions with an execution time more than 3 orders of magnitude faster in networks up to 24 nodes. Moreover, we discussed the potential applications that can have this GNN-based explainability solution when applied to networking. As future work, it would be interesting to show experimentally the potential applications of the proposed lightweight explainability method to different networking use cases, such as those described in Section 7, as well as making a deep analysis on the knowledge extracted by NetXplain on different target GNN models.

ACKNOWLEDGEMENT

This work has been supported by the Spanish MINECO under contract TEC2017-90034-C2-1-R (ALLIANCE) and the Catalan Institution for Research and Advanced Studies (ICREA).

REFERENCES

[1] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[2] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[3] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. "Interpreting Deep Learning-Based Networking Systems". In: *Proceedings of ACM SIGCOMM*. 2020, pp. 154–171.

[4] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models". In: *arXiv preprint arXiv:1708.08296* (2017).

[5] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. "Network dissection: Quantifying interpretability of deep visual representations". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 6541–6549.

[6] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20.1 (2008), pp. 61–80.

[7] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. "Neural message passing for quantum chemistry". In: *arXiv preprint arXiv:1704.01212* (2017).

[8] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. "Interaction networks for learning about objects, relations and physics". In: *Advances in neural information processing systems (NIPS)*. 2016, pp. 4502–4510.

[9] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. "Modeling polypharmacy side effects with graph convolutional networks". In: *Bioinformatics* 34.13 (2018), pp. i457–i466.

[10] Rex Ying et al. "Graph convolutional neural networks for web-scale recommender systems". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 974–983.

[11] Wenqi Fan et al. "Graph neural networks for social recommendation". In: *The ACM World Wide Web Conference (WWW)*. 2019, pp. 417–426.

[12] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN". In: *Proceedings of the 2019 ACM Symposium on SDN Research*. 2019, pp. 140–151.

[13] Fabien Geyer and Georg Carle. "Learning and generating distributed routing protocols using graph-based deep learning". In: *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. 2018, pp. 40–45.

[14] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. "Learning scheduling algorithms for data processing clusters". In: *Proceedings of the ACM SIGCOMM*. 2019, pp. 270–288.

[15] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case". In: *arXiv preprint arXiv:1910.07421* (2019).

[16] Arnau Badia-Sampera, José Suárez-Varela, Paul Almasan, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Towards more realistic network models based on Graph Neural Networks". In: *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*. 2019, pp. 14–16.

[17] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. "Gnnexplainer: Generating explanations for graph neural networks". In: *Advances in neural information processing systems*. 2019, pp. 9244–9255.

[18] *Network Modeling Datasets*. https://github.com/knowledgedefinednetworking/NetworkModelingDatasets. Jan. 2020.

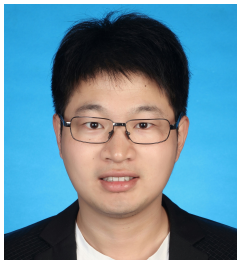
[19] Steven Gay, Renaud Hartert, and Stefano Vissicchio. "Expect the unexpected: Sub-second optimization for segment routing". In: *IEEE INFOCOM 2017*. 2017, pp. 1–9.

AUTHORS

David Pujol-Perich is an MSc student in advanced computing from UPC (2020-2022), where he is also received his Bachelor's degree in computer science (2016-2020). He is currently employed by Barcelona Neural Networking center (BNN-UPC) as a machine learning junior researcher. During his Bachelor's, he did an academic stay in ETH Zürich (2019-2020) and was awarded as finalist of the Best Bachelor Thesis of the Year.
In addition, the project resulting from his bachelor thesis, IGNNITION, received an EU grant (H2020-871528-NGI-POINTER; 1st open call). His research focuses on the development of tools that allow non-expert users in ML to easily implement complex graph neural networks. Furthermore, he is also working on novel techniques to explain the knowledge that graph neural networks automatically acquire to accurately make predictions.

José Suárez-Varela received his M.Sc. degree in telecommunication engineering from the Universidad de Granada (UGR) in 2017, and his Ph.D. from the Universitat Politècnica de Catalunya (UPC) in 2020. He is currently a post-doctoral researcher at the Barcelona Neural Networking Center (BNN-UPC), and co-Principal Investigator of the EU-funded project IGNNITION (H2020-871528-NGI-POINTER; 1st open call).
During his Ph.D., he was a visiting researcher at the University of Siena, under the supervision of Prof. Franco Scarselli. His main research interests are in the field of artificial intelligence applied to networking, and traffic monitoring and analysis.

Shihan Xiao received a B.Eng. degree in electronic and information engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2012, and a Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, China. He is currently a senior engineer with Huawei 2012 NetLab. His research interests include machine learning in networking, data center networking, and cloud computing.



Bo Wu received his Bachelor's degree from the School of Software at Shandong University, China, in 2014, and his Ph.D. degree from the Department of Computer Science and Technology at Tsinghua University in 2019. Currently, he works in the Network Technology Laboratory at Huawei Technologies. His research interests include network architecture, network security, and blockchain.

Albert Cabellos-Aparicio is an assistant professor at Universitat Politecnica de Catalunya (UPC), where he obtained his PhD in computer science in 2008. He is director of the Barcelona Neural Networking Center (BNN-UPC) and scientific director of the NaNoNetworking Center in Catalunya. He has been a visiting researcher at Cisco Systems and Agilent Technologies and a visiting professor at KTH, Sweden, and MIT, USA. His research interests include the application of machine learning to networking and nanocommunications. His research achievements have been awarded by the Catalan Government, the University, and INTEL. He also participates regularly in standardization bodies such as IETF.

Pere Barlet-Ros is an associate professor at Universitat Politècnica de Catalunya (UPC) and scientific director of the Barcelona Neural Networking Center (BNN-UPC). From 2013 to 2018, he was co-founder and chairman of the machine learning startup Talaia Networks, which was acquired by Auvik Networks in 2018. He was a visiting researcher at Endace (New Zealand), Intel Research Cambridge (UK) and Intel Labs Berkeley (USA). His research interests are in machine learning for network management and optimization, traffic classification and network security. He received the 2nd VALORTEC prize (2014) for the best business plan awarded by the Catalan Government (ACCIO).